Budapest University of Technology and Economics
Department of Telecommunication and Telematics

Conformance Test Suite Optimization

PhD thesis

Tibor Csöndes

Advisors

Dr. Katalin Tarnay
dr. Gyula Csopaki
dr. Sarolta Dibuz

Budapest, Hungary
2001

Abstract

This thesis presents a mathematical model for test selection problem in *protocol conformance testing*, the goal of which is to select a test set from a given test suite. Finding a suitable test set usually leads to optimization problems. The problem is described together with its mathematical formulation including two *optimization problems* and four different *coverage models*. We also transform the test selection problem to *Integer Programming* (IP) problem and it is shown that our problem is NP-hard.

We propose four simple greedy algorithms and a more complex one for the selection, which are able to model the manual test selection and can provide good solutions as the experimental results show. We also apply different meta-heuristics, namely Reactive Tabu Search (RTS), Genetic Algorithms (GA) and Simulated Annealing (SA) to solve the problem. We propose some modifications to the conventional schemes including a new adaptive neighbourhood sampling in RTS, adaptive variable mutation rate in GA and an adaptive variable neighbourhood structure in SA.

The performance of the algorithms is evaluated in different models for existing protocol implementations. Computational results show that GA and SA can provide high-quality solutions in acceptable time compared to the result of a commercial software, which makes them applicable in practical test selection.

Finally, we present the process of manual and automatic test selection and apply our mathematical apparatus to real-life applications.

Acknowledgements

First of all, I would like to thank my advisors, Dr. Katalin Tarnay, for her authoritative guidance in finding the proper scientific direction, dr. Gyula Csopaki, for his valuable help during the finalization of my thesis, and dr. Sarolta Dibuz, who gave me opportunity to carry out my research at the Conformance Laboratory at Ericsson Hungary.

I would like to thank my colleagues, especially Balázs Kotnyek, who helped me continuously during the whole research project and who co-authored several of my scientific papers; Tibor Cinkler and Alpár Jüttner, for their original ideas in the field of operational research, and János Zoltán Szabó, for his valuable help in computer programming.

Last, but not least, I would like to thank Pál Kovács, who introduced me first to conformance testing.

Contents

Abstract ii

Acknowledgements iii

Contents iv

List of Figures vii

1 Introduction 1

1.1 Conformance testing . . . 1

1.1.1 The process of conformance testing . . . 2

1.1.2 Conformance requirements . . . 3

1.1.3 Test purposes, test cases . . . 3

1.1.4 The Abstract Test Suite . . . 4

1.1.5 Tree and Tabular Combined Notation (TTCN) . . . 5

1.2 Test generation . . . 9

1.2.1 Transition Tour method . . . 11

1.2.2 Distinguishing Sequence method . . . 12

1.2.3 W-method . . . 13

1.2.4 Unique Input/Output (UIO) sequences method . . . 14

1.3 Non-FSM-based conformance testing methodologies . . . 15

1.3.1 Labelled Transition Systems . . . 15

1.3.2 Metric Based Method . . . 16

1.4 Test selection . . . 16

1.4.1 Selecting from a generated test set . . . 17

1.4.2 Selecting from the ATS . . . 17

2 Automation in test selection problem **19**

2.1 Definition of subpurposes 19

2.2 Determination of subpurposes 22

2.3 Abstract data levels 23

3 Mathematical model for test selection **25**

3.1 Problem formulation 25

3.2 Coverage models 27

4 Test selection process **30**

4.1 Creation of the subpurpose-test case incidence matrix 31

4.1.1 Manual matrix filling 31

4.1.2 Automatic matrix filling 32

4.2 Creation of cost and weight vectors 33

5 Mathematical solution methods **34**

5.1 ILP formulation 35

5.2 Heuristics 38

5.2.1 Greedy algorithms 40

5.2.2 Tabu Search 42

5.2.3 Genetic Algorithms 45

5.2.4 Simulated Annealing 48

6 Parameter settings in the solution methods **51**

6.1 Branch and Bound 52

6.2 Tabu Search 53

6.3 Genetic Algorithm 54

6.4 Simulated Annealing 55

6.4.1 Cooling schedule 55

6.4.2 Neighbourhood structure 58

7 Experimental results on heuristics **59**

7.1 Greedy algorithms 59

7.1.1 Results of the greedy algorithms 60

7.1.2 Examining the results 60

7.2 Reactive Tabu Search, Genetic Algorithms and Simulated Annealing 61

7.2.1 Results of the Reactive Tabu Search, Genetic Algorithms and Simulated Annealing . . . 61
7.2.2 Examining the results . . . 62

8 Real-life applications 64
8.1 ETSI TBR4 application . . . 65
8.1.1 Minimal cost problems . . . 65
8.1.2 Maximal coverage problems . . . 66
8.2 Function test application . . . 66
8.2.1 Input data and coverage models . . . 67
8.2.2 Results of the test selection . . . 67

9 Conclusions 70
9.1 Summary of the thesis . . . 70
9.2 Further research . . . 71

A Additional results on heuristics 73

B Additional results on ETSI TBR4 application 77

C Additional results on function test application 81

Abbreviations 84

Glossary of symbols 86

Bibliography 89

List of Figures

1.1 The hierarchical structure of the ATS 5
1.2 Tree notation and tabular format 6
1.3 TTCN.GR and TTCN.MP forms 6
1.4 FSM representation with a directed graph 10
1.5 Characterizing set (W-method) example 13

2.1 Requirements, test purposes and subpurposes 21

3.1 Subpurpose-test case incidence matrix 26
3.2 Coverage models 28

4.1 The process of selection 30
4.2 An example for a requirement and the related test cases in ETSI TBR4 . . 32

5.1 The adaptive variable mutation rate in GA 47
5.2 The adaptive variable neighbourhood structure in SA 49

6.1 The average solutions and their projections 55
6.2 The graph of the average execution times and its projection 56
6.3 The efficiency function and its projection 57
6.4 The standard deviations of the solutions and their projections 57

A.1 Greedy algorithms for GSM in minimal cost problem, One is Enough model 73
A.2 Greedy algorithms for GSM in minimal cost problem, Step and Jump model 73
A.3 Greedy algorithms for GSM in minimal cost problem, All or Nothing model 73
A.4 Greedy algorithms for INAP in minimal cost problem, One is Enough model 73
A.5 Greedy algorithms for INAP in minimal cost problem, Step and Jump model 74
A.6 Greedy algorithms for INAP in minimal cost problem, All or Nothing model 74

B.1 Minimal cost problem, All or Nothing model, $\mathbf{c_1}$ cost vector 77
B.2 Minimal cost problem, All or Nothing model, $\mathbf{c_2}$ cost vector 77
B.3 Minimal cost problem, All or Nothing model, $\mathbf{c_3}$ cost vector 78
B.4 Minimal cost problem, Linear model, $\mathbf{c_1}$ cost vector 78
B.5 Minimal cost problem, Linear model, $\mathbf{c_2}$ cost vector 78
B.6 Minimal cost problem, Linear model, $\mathbf{c_3}$ cost vector 78
B.7 Minimal cost problem, One is Enough model, $\mathbf{c_1}$ cost vector 78
B.8 Minimal cost problem, One is Enough model, $\mathbf{c_2}$ cost vector 78
B.9 Minimal cost problem, One is Enough model, $\mathbf{c_3}$ cost vector 79
B.10 Maximal coverage problem, All or Nothing model, $\mathbf{c_1}$ cost vector 79
B.11 Maximal coverage problem, All or Nothing model, $\mathbf{c_2}$ cost vector 79
B.12 Maximal coverage problem, All or Nothing model, $\mathbf{c_3}$ cost vector 79
B.13 Maximal coverage problem, Linear model, $\mathbf{c_1}$ cost vector 79
B.14 Maximal coverage problem, Linear model, $\mathbf{c_2}$ cost vector 79
B.15 Maximal coverage problem, Linear model, $\mathbf{c_3}$ cost vector 80
B.16 Maximal coverage problem, One is Enough model, $\mathbf{c_1}$ cost vector 80
B.17 Maximal coverage problem, One is Enough model, $\mathbf{c_2}$ cost vector 80
B.18 Maximal coverage problem, One is Enough model, $\mathbf{c_3}$ cost vector 80

C.1 Minimal cost problem, Linear model, $\mathbf{w_1}$ weight vector 81
C.2 Minimal cost problem, Linear model, $\mathbf{w_2}$ weight vector 81
C.3 Minimal cost problem, Linear model, $\mathbf{w_3}$ weight vector 82
C.4 Minimal cost problem, One is Enough model, $\mathbf{w_1}$ weight vector 82
C.5 Minimal cost problem, One is Enough model, $\mathbf{w_2}$ weight vector 82
C.6 Minimal cost problem, One is Enough model, $\mathbf{w_3}$ weight vector 82
C.7 Maximal coverage problem, Linear model, $\mathbf{w_1}$ weight vector 82
C.8 Maximal coverage problem, Linear model, $\mathbf{w_2}$ weight vector 82
C.9 Maximal coverage problem, Linear model, $\mathbf{w_3}$ weight vector 83
C.10 Maximal coverage problem, One is Enough model, $\mathbf{w_1}$ weight vector 83
C.11 Maximal coverage problem, One is Enough model, $\mathbf{w_2}$ weight vector 83
C.12 Maximal coverage problem, One is Enough model, $\mathbf{w_3}$ weight vector 83

Chapter 1

Introduction

The purpose of this thesis is to present a theoretical test selection method which besides its theoretical soundness also keeps the practical aspects in view. A good test selection has important economic advantages. Test laboratories can complete testing faster or they can achieve higher coverage in a given time. Therefore, a method which can help them in making the selection more efficient has high practical significance.

Reducing the time or the efforts put into conformance testing while keeping the given confidence level is very important for those who perform conformance testing, usually in a test laboratory. If the time for conformance testing is limited, testers need to select the most efficient test cases from the whole test suite in order to make testing feasible within a shorter period of time.

Our approach to solve this problem is based on selecting test cases from the Abstract Test Suite of a protocol, where we can select what portion of the test coverage we are willing to devote to shorten the time of test execution. To achieve minimal testing time for a given lower bound of coverage or maximal coverage for an upper bound of the cost, the method determines which test cases have to be selected for execution. The method is protocol independent, so it can be used in the testing of any protocol implementation.

As a background to our method, in this chapter we introduce the most important and relevant notions of conformance testing.

1.1 Conformance testing

Conformance testing is an important step in the life-cycle of telecommunication protocols ([Baum94]). Its purpose is to check whether the protocol implementation is conform to

standards, and by that to increase the probability that different implementations will be able to interwork. Conformance testing involves verifying that the external behaviour of the implementations comply with the requirements contained in their relevant protocol specifications ([Bush90]). In order to get a reliable and repeatable verdict of the conformance, a standardized test suite, an *Abstract Test Suite* (ATS) has to be used. As the protocols, the Abstract Test Suites and the test process are standardized by international standardization institutes, the International Organization for Standardization (ISO), the International Telecommunication Union - Telecommunication standardization sector (ITU-T), and the European Telecommunications Standards Institute (ETSI). Conformance testing and its methodology are based on the standard [ISO9646]. We will summarize this methodology in this Introduction.

1.1.1 The process of conformance testing

According to the standard, the conformance testing assessment process is the complete process of testing activities necessary to assess the protocol implementation's conformance. The conformance assessment process involves three main phases ([ISO9646, Baum94, Tretmans92]):

- Preparation for testing
- Test operation
- Test report production

In the **preparation** phase, the implementors have to make a statement to determine which capabilities their *Implementation Under Test* (IUT) does support, and have to fill in the *Protocol Implementation Conformance Statement* (PICS) document stating the optional features they have selected from the protocol standard to include in their product. In collaboration with the test laboratory, they supply a *Protocol Implementation eXtra Information for Testing* (PIXIT) document as a mean for providing any non-standard parameter values necessary to carry out the testing process.

The **test operation** involves the static conformance review of the PICS, checking the consistency of the PIXIT, the test selection[1] and parameterization based on the PICS and the PIXIT documents, and the test campaigns.

[1] Note that the term "test selection" in the context of ISO9646 refers to the selection process of test cases from an abstract test suite based on values of PICS and the PIXIT. The definition of "test selection" that we will use is different. It is discussed in Section 1.4.

During the **test report production,** the process is documented in a *Protocol Conformance Test Report* (PCTR).

International standardization institutes produce testing documents to standardize the conformance testing process of different protocols. Moreover, they issue standardized test documents such as ATS, PICS, PIXIT, PCTR, and test purpose documents for several important protocols.

1.1.2 Conformance requirements

During conformance testing, a protocol implementation is tested to ensure that it meets the *conformance requirements* contained informally in the relevant protocol specification ([Bush90]). Basically, a protocol specification is made up of the conformance requirements.

As the standard ISO 9646 states, a *conforming implementation* is "an IUT which satisfied both static and dynamic conformance requirements, consistent with the capabilities stated in the PICS(s)". A *static conformance requirement* is "one of the requirements that specify the limitations on the combinations of implemented capabilities permitted in a real open system which is claimed to conform to the relevant specification(s)", and a *dynamic conformance requirement* is "one of the requirements which specifies what observable behaviour is permitted by the relevant specification(s) in instances of communication" ([ISO9646]).

As these descriptions of the standard imply, a correct implementation is defined as one which satisfies all conformance requirements. Testing of the static conformance requirements simply means a reviewing process of the PICS delivered with the IUT, called static conformance review. Testing dynamic conformance requirements involves running a number of test cases against the IUT, called *test campaign* ([Tretmans92]). As checking the static conformance requirements is simple and does not bring about any theoretical problem, we will deal only with the dynamic conformance requirements in this thesis.

1.1.3 Test purposes, test cases

The standard defines a *Test Purpose* (TP) as "a prose description of a well defined objective of testing, focusing on a single conformance requirement or a set of related conformance requirements as specified in the appropriate OSI specification (e.g.: verifying the support of a specific value of a specific parameter)" ([ISO9646]).

Standardization institutes collect the conformance requirements of protocol specification and produce test purpose documents. These standardized documents contain several test purposes in a well defined form. Different standardization institutes use different forms. For example, within ETSI, this task is described in the "Test Purpose style guide" ([ETR266]), showing the method for document writers to prepare a test purpose document.

A test purpose is supposed to be based on a single conformance requirement. In practice, however, test purposes may concern a set of conformance requirements. The "Test Purpose style guide" lists the following cases when this is allowed:

- when the test case concerns an aspect specific to a profile;
- when the size of the ATS demands a limitation in the number of test purposes ([ETR266]).

Based on the test purpose document, *Test Cases* (TCs) are derived: one test case for each test purpose. This derivation can be done automatically or manually. The automatic test case derivation is based on test generation methods, described in Section 1.2. The manual test case writing focuses on the protocol specification and the informally given conformance requirements collected in the test purpose document.

1.1.4 The Abstract Test Suite

The result of the test writing process is the Abstract Test Suite (ATS). The ATS contains a considerable number of test cases, each of which focuses on a specific part of the protocols' behaviour described in its purpose, and which can be executed independently from each other. The test suite is called abstract because it is independent of the implementation. This implies that an ATS is not directly executable, as it needs further parameterization and compilation.

Ideally, an ATS should be *complete*, that is, an implementation passes it if and only if the implementation is correct. Since, it is generally not possible to construct a finite complete test suite, an ATS is required to be *sound,* meaning that all implementations that do not pass it are not correct. A sound test suite is complete if and only if it is also *exhaustive*; that is, if all passing implementations are conforming ([FMCT]).

The ATS has a hierarchical structure. The central component is the test case, which is grouped with other test cases into *test groups.* Test cases are decomposed into test steps and test steps are decomposed into test events ([ISO9646], [Probert92]). The hierarchical structure of the ATS is illustrated in Figure 1.1.

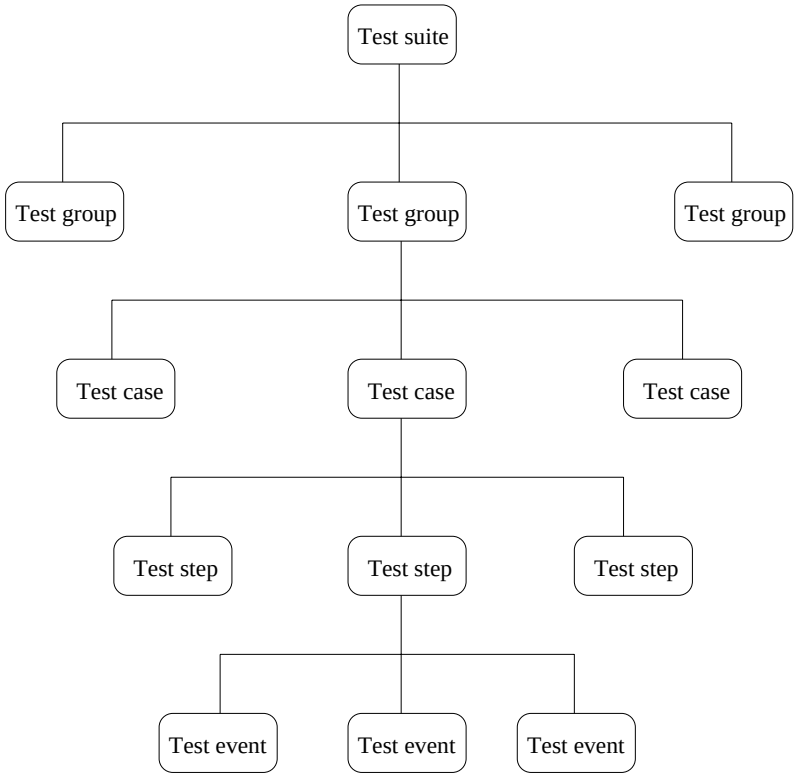

Figure 1.1: The hierarchical structure of the ATS

Test suites must be specified according to a test notation. A good test notation is well-defined, independent of the implementation, and is generally accepted. The standard ISO 9646 recommends a semi-formal language, the Tree and Tabular Combined Notation (TTCN).

1.1.5 Tree and Tabular Combined Notation (TTCN)

During the standardization of the TTCN, theoretical experts of the test suite generation methods proposed a behaviour tree notation, while manufacturers suggested a tabular format. The advantage of using test trees is that the dynamic behaviour of the test execution becomes simply traceable, whereas with the aid of the tabular format it is easy to specify the testing objects unambiguously. Finally, a compromise was made and the tabular format was mixed with the behaviour tree notation by creating the Tree and Tabular Combined Notation ([ISO9646]). Figure 1.2 shows a simple example in both tree and tabular formats.

TTCN can be given in two forms: a *graphical form,* denoted as TTCN.GR, and a *machine-processable form*, denoted as TTCN.MP. The graphical form is defined using

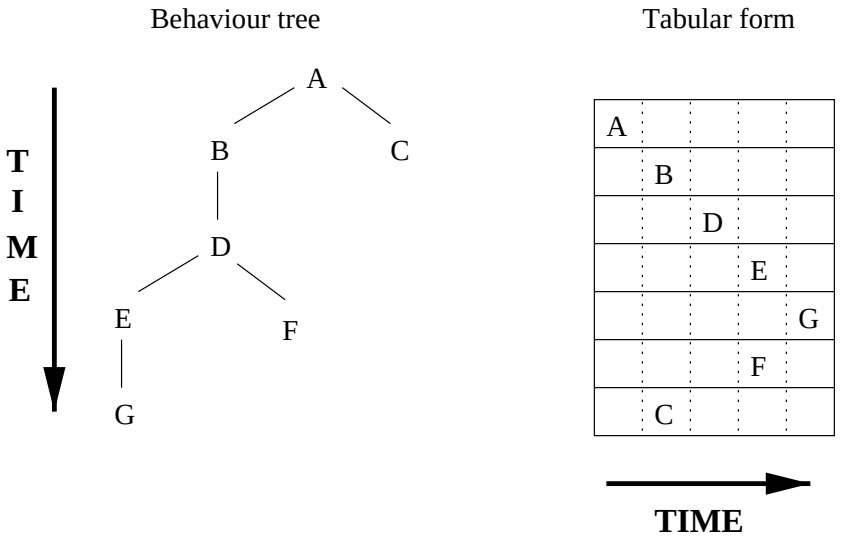

Figure 1.2: Tree notation and tabular format

tabular proformas and it is suitable for human reading or visual interpretation and it is easy to understand. TTCN.MP is suitable for exchange of TTCN descriptions between different machines and applications or for other automated processing. That is, TTCN is defined in such a way that its automatic execution is possible. In fact, the two different representations of an ATS in graphical and machine processable format are equivalent. Moreover, using a commercial TTCN editor it is easy to convert a TTCN.GR to TTCN.MP and vice versa ([ISO9646, Baum94, Probert92]). Figure 1.3 shows a simple example of the table and machine processable forms.

TTCN.GR form

Test Case Dynamic Behaviour

Test Case Name: Dynamic_behaviour_example
Group:
Purpose: Clarify behaviour tree concept
Configuration:
Default:
Comments: Reference 5.5.1.7 / PICS

Nr	Lab.	Behaviour description	Const. Ref.	Verdict	Comments
1		A			
2		B			
3		D			

TTCN.MP form

```
$ Begin_TestCase
$ TestCaseId Dynamic_behaviour_example
$ TestGroupRef
$ TestPurpose /* Clarify behaviour tree concept */
$ DefaultsRef
$ ConfigurationRef
$ Comment /* Reference 5.5.1.7 / PICS */
$ SelectExprId
$ Description /* */
$ BehaviourDescription
$ BehaviourLine
$ LabelId
$ Line [0] A
$ Cref
$ VerdictId
$ Comment /* */
$ End_BehaviourLine
...
```

Figure 1.3: TTCN.GR and TTCN.MP forms

A TTCN test suite consists of four main sections:

- *Test suite overview part*
- *Declarations part*
- *Constraints part*
- *Dynamic behaviour part*

Test suite overview part

The test suite overview part first names the test suite and refers to the appropriate protocol specification, PICS, PIXIT documents, etc. This part includes a commented overview of the test suite and contains commented indices of the test cases ([Baum94]).

Declarations part

The purpose of the declarations part of the ATS is to define and declare all the objects used in the test suite ([ISO9646]). The most important objects in the declarations part are:

- User defined types
- User defined operations
- Test suite parameters (constants derived from the PICS and/or PIXIT)
- Global and local constants
- Global and local variables
- Abstract Service Primitives (ASPs)
- Protocol Data Units (PDUs)
- Timers
- Abbreviations

OSI protocol specifications define the allowed behaviour of a protocol implementation (i.e.: the dynamic conformance requirements) in terms of *Protocol Data Unit*s and *Abstract Service Primitive*s ([ISO9646]).

Constraints part

An ATS has to specify the values of ASP parameters and PDU fields that are sent and received by the test system. The constraint part fulfills this purpose in TTCN ([ISO9646]). In the constraints part, particular data values are specified for PDU fields and ASP parameters as they are used in a constrained test event in the dynamic behaviour part and declared in the declarations part. This allows the specification of test events to be short and simple ([Probert92]).

Constraints may be parameterized. In this case the names of the constraints are followed by a parenthesized formal parameter list and the parameterized ASP parameters and PDU fields will have these parameters as values ([Sarikaya92]).

Dynamic behaviour part

The dynamic behaviour part contains the main body of the test suite, the behaviour descriptions (i.e.: the real tests) ([ISO9646]). Test cases show the behaviour trees in tabular format (Figure 1.2). A test case dynamic behaviour table consists of test steps, *defaults,* and test events referring to the ASPs, PDUs together with their parameters, which can be found in the constraints part.

Test steps and defaults help to modularize the test suite. Test steps contain those series of test events that are used in several test cases, making the description of test cases more compact. Defaults are very similar to test steps but specify alternative behaviour for all not specified events. In a behaviour tree it often happens that every sequence of alternatives ends in the same way. The default may contain this common behaviour.

The ultimate purpose of conformance testing is to assign verdicts to test cases. Every test case execution must result in a verdict, which is either *pass*, *fail*, or *inconclusive*:

- **Pass**, if the behaviour of the IUT conforms to the related protocol specification
- **Fail**, if the behaviour of the IUT is faulty; if it does not conform to the related protocol specification
- **Inconclusive**, if the verdict is neither pass nor fail; if we cannot decide whether the IUT conforms

1.2 Test generation

Large improvements in telecommunication technology have resulted in an increasing demand of powerful testing and efficient test suites. Therefore, the automatic test suite generation methods came to the forefront of research and several scientific papers were published in this topic. The results were very promising, but their practical importance were (and still are) questionable. It is not easy to put the test generation algorithms into practice, mainly because they need a formal description of the protocol behaviour, which is hardly available. Furthermore, the size of generated test suites often go out of control. In this section we will summarize the important test generation methods and point out some of their advantages and disadvantages.

The goal of the test generation methods, in summary, is to generate ATSs that are optimal, that is, they contain as few redundancies as possible. The process usually starts with the protocol specification given as a *Finite State Machine* (FSM).

Definition 1. ([Kohavi78, Uyar98]) A deterministic Finite State Machine is a 6-tuple $FSM = \langle S, I, O, \delta, \theta, st_0 \rangle$ where

- $S = \{st_1, \ldots, st_n\}$ is a finite set of states
- $I = \{i_1, \ldots, i_n\}$ is a finite set of inputs
- $O = \{o_1, \ldots, o_n\}$ is a finite set of outputs
- δ : a state transition function that maps the current state and input to the next state: $\delta : S \times I \longrightarrow S$
- θ : an output function that maps the current state and input to the output: $\theta : S \times I \longrightarrow O$
- st_0 : the initial state of the FSM, which is the state immediately after power-up

An FSM can be represented by a *directed graph* or a *state-transition table* ([Holzmann91]). In the graph representation of the FSM, the vertices of the graph correspond to the states. A directed edge from st_i to st_j with label i_k/o_l represents a state transition from state st_i to state st_j by applying input i_k and observing output o_l ([Bosik91, Uyar98]). We will denote this edge as $\{st_i, st_j, i_k/o_l\}$. An example for a graph representation can be seen in Figure 1.4.

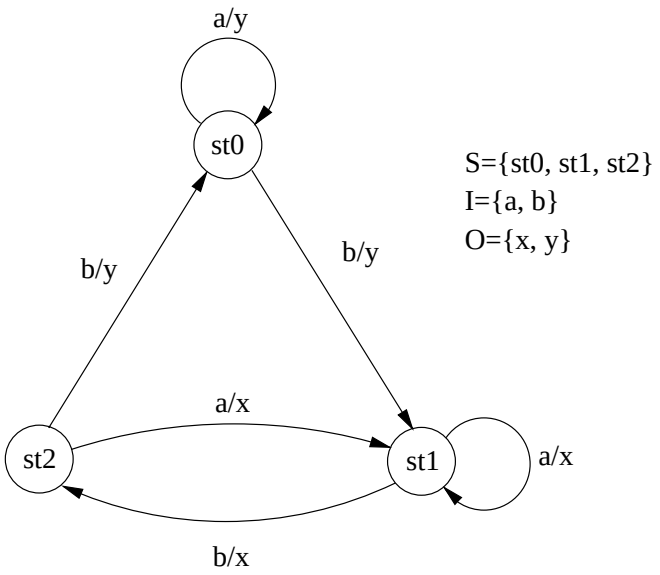

Figure 1.4: FSM representation with a directed graph

We can also represent an FSM with a state-transition table. The rows of the table correspond to the states of the FSM, while the columns correspond to the inputs. Each entry of the table shows the next state and the output, produced when the input of the column is applied to the state of the row. Table 1.1 shows the state-transition table representation of the FSM of Figure 1.4.

Table 1.1: FSM representation with a state-transition table

S\I	a	b
st0	st0/y	st1/y
st1	st1/x	st2/x
st2	st1/x	st0/y

The behaviour of a protocol can also be described with an *Extended Finite State Machine* (EFSM). An EFSM is a Finite State Machine where the states can be additionally associated with conditional statement or variable assignments. The EFSMs are compact versions of FSMs ([Bosik91]). During the conversion of an EFSM to FSMs the problem of *state-space explosion* can occur, because a state of the EFSM can result in an astronomically large number of corresponding states in the FSM. This is a serious problem, as real-life protocols are modeled only with EFSM, whereas the test generation methods work mostly on FSMs.

If an FSM model of the protocol specification exists, there are several algorithms for generating good test suites. The most important algorithms are:

- Transition Tour (TT) method
- Distinguishing Sequence (DS) method
- W-method
- Unique Input/Output (UIO) sequences method

The test generation methods assume the *black box* approach, where the implementation has an input port and an output port, and one can control only the inputs generated by the testing device and observe the outputs generated by the implementation, but not the state of the implementation ([Dahbura90, Sabnani88]),. whereas testing an implementation modeled by an FSM means checking all the state transitions.

Testing a state transition from state st_i to st_j with the input-output operation i_k/o_l takes three steps ([Bosik91, Dahbura90]):

$$\begin{array}{l} \text{Step 1: Put the FSM implementation into state } st_i \\ \text{Step 2: Apply input } i_k \text{ and observe output } o_l \\ \text{Step 3: Verify that the new state of FSM is } st_j, \text{ as expected} \end{array} \tag{1.1}$$

Each FSM-based method assumes that the graph representation of the FSM implementation is *strongly-connected,* meaning that it is possible to reach any specified state from any other state via a sequence of valid inputs. In other words, the formal test generation methods assume to be verified that the implementation does not contain deadlocks and live-locks ([Bosik91, Dahbura90]).

Finally, it is required that the generated test suite is sound, that is, the generated test cases represent correct behaviour.

1.2.1 Transition Tour method

This method is the simplest among the ones discussed here. The Transition Tour (TT) method was first published in [Naito81] for sequential circuits. The first application for protocol testing is found in [Sarikaya82]. The method requires that the implementation modeled by an FSM is minimal, strongly-connected, and fully-specified. The test sequence

the TT method produces can detect all operation errors (errors of the output), but it does not detect all transfer errors (errors of the next state). In other words, this method does not execute Step 3 of the test procedure (1.1). For this reason, the Transition Tour method can be used for test generation only if the protocol specification contains a "status" message which can uniquely identify every states in the FSM.

The simplicity of the Transition Tour method arises from that it generates a sequence of inputs and outputs (called a *tour*) that exercises every state transition of the implementation. A tour is optimal if it covers every state transition of an FSM specification and requires a minimum number of state transitions. The optimal tour generation is based on a graph theory problem called *Chinese Postman Problem.* The Chinese Postman Problem is to find a tour of a graph that starts and ends at the same vertex, at minimum cost, and that traverses each edge of the graph at least once [Bosik91]).

We illustrate the method on the example FSM of Figure 1.4. The optimal Transition Tour is as follows:

I/O:	a/y	b/y	a/x	b/x	a/x	b/x	b/y	
State:	st0	st0	st1	st1	st2	st1	st2	st0

As stated above, unless the protocol specification contains a "status" message, the TT method is not suitable for complete test generation because it does not execute Step 3 of the test procedure (1.1). The following three methods were invented to solve this problem as they can verify the new state after a transition (i.e.: they execute Step 3 of (1.1)). Together with the Transition Tour, they provide us with a complete test generation method.

1.2.2 Distinguishing Sequence method

This method assumes that the FSM is minimal, strongly-connected and fully-specified. An input string x is said to be a *Distinguishing Sequence* of a machine M, if the output string produced by M in response to x is different for each starting state. An algorithm to find a Distinguishing Sequence in a Finite State Machine was presented in [Gonenc70].

There is no Distinguishing Sequence for every FSM ([Sidhu89]), hence this method is not suitable for all protocols. A Distinguishing Sequence of the FSM depicted in Figure 1.4 is $\{ba\}$. The output sequences for different starting states are as follows:

DS= ba

Starting state		Output
st_0	$\longrightarrow$	yx
st_1	$\longrightarrow$	xx
st_2	$\longrightarrow$	yy

The major drawback of this method is that in case of real-life protocols very few FSMs have a Distinguishing Sequence ([Bosik91]).

1.2.3 W-method

For the FSMs that do not have Distinguishing Sequences the W-method can be applied. The W-method assumes a minimal, strongly-connected, and fully-specified machine. It involves deriving a *characterization set* W of the FSM, consisting of *characterization sequences* ([Sidhu89]). A characterizing sequence is a sequence of inputs that is able to distinguish state st_i from a subset of the other states.

As a characterization set exists for any FSM, the W-method is more applicable to real-life implementation than the DS method ([Bosik91]). As an example, let us change the label of the directed edge from st_1 to st_2 in Figure 1.4 from b/x to b/y as in Figure 1.5. It is easy to verify that this FSM does not have a Distinguishing Sequence, while, as we will show, it does possess a characterization set.

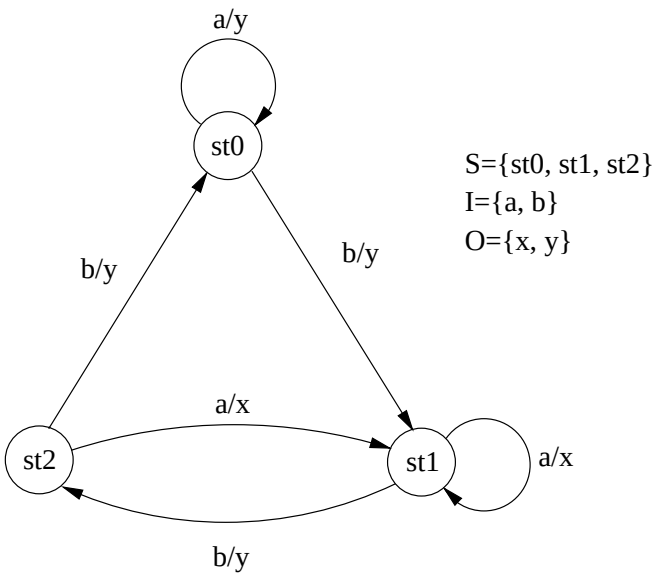

Figure 1.5: Characterizing set (W-method) example

Now we derive characterizing sequences for the FSM showed in Figure 1.5. First, we distinguish state st_0 from the others:

Characterizing sequence $1 = a$

Starting state		Output
st_0	$\longrightarrow$	y
st_1	$\longrightarrow$	x
st_2	$\longrightarrow$	x

Then we have to distinguish state st_1 and st_2 from each other:

Characterizing sequence $2 = ba$

Starting state		Output
st_0	$\longrightarrow$	yx
st_1	$\longrightarrow$	yx
st_2	$\longrightarrow$	yy

The result of the W-method is the characterization set $W = \{a, ba\}$. An algorithm to find a characterization set is found in [Chow78].

1.2.4 Unique Input/Output (UIO) sequences method

For verifying the new state of a state transition, the UIO method compute a unique signature for each state. A *UIO sequence* for a state is an input/output behaviour that is not produced by any other state.

The UIO method is different from the Distinguishing Sequence method because in the latter one, the input sequence is the same for all states, while in the UIO method the input sequence can differ for different states. UIO sequences are also different from the characterizing sequences. The input sequence of the W-method can distinguish a state from the subset of other states, while the UIO sequence can distinguish a state from all other states ([Sabnani88]).

The main difference is, however, that the UIO sequence for a state can only verify that the FSM was at state st_i before the UIO was applied, and does not identify the current state of the FSM. In other words, the DS and W-methods are able to answer the following question:

"What is the state of the FSM?"

Whereas the UIO method only answers the question:

"Was the implementation in state st_i?"

The UIO sequence method is preferable to the other methods because:

- almost all FSMs have UIO sequences, while only a few have a Distinguishing Sequence
- the UIO sequences are typically much shorter than the DS or the characterizing sequences

For the application of the UIO method, a minimal, strongly connected and fully specified FSM is required. To find minimal cost UIO sequences, Linear Programming can be used ([Bosik91]). The UIO sequences for the states of the FSM depicted in Figure 1.4 are the following:

Starting state		UIO of the state
st_0	$\longrightarrow$	a/y
st_1	$\longrightarrow$	b/x
st_2	$\longrightarrow$	b/y, a/y

A *UIO set* for an FSM is the generalization of the UIO sequence similarly to the way as the W-set generalizes the DS method. There are FSMs for which the UIO sequences of certain states do not exist, when a set of partial UIO sequences is used ([Sun97]).

1.3 Non-FSM-based conformance testing methodologies

Usually, protocol specifications are given in the formalism of input/output finite state machines (FSMs) or formal languages based on FSMs. However, there are also formal description techniques for protocol specifications, such as LOTOS ([Brinksma87, ISO8807]) and CCS ([Milner80]), that are based on the so-called Labelled Transition Systems (LTS), rather than FSMs; or execution sequences which are the basis of the Metric Based Method ([Vuong92, Vuong97, Zhu97]).

1.3.1 Labelled Transition Systems

Definition 2. ([Tretmans92]) A labelled transition system is a 4-tuple $\langle S, L, T, s_0 \rangle$ where

- S is a (countable) non-empty set of states

- L is a (countable) set of observable actions
- $T \subseteq S \times (L \cup \{\tau\}) \times S$ is the transition relation
- $s_0 \in S$ is the initial state

The labels in L represent the observable interactions of a system; the special label $\tau \notin L$ represents an unobservable, internal action.

A framework for LTS based testing is presented in [Tan97]. There are two approaches used to derive test suites from an LTS: the direct test derivation and the translation to FSM. In the latter one, a given LTS specification is translated to an FSM using conformance relations such as trace equivalence, failure reduction, nondeterministic reduction and failure equivalence; an FSM-based method (like the UIO, DS or W-method) is applied and the obtained FSM test suite is converted to an LTS test suite.

1.3.2 Metric Based Method

In this method a metric space is defined over the behaviour space of the protocol made of *execution sequences*. An execution sequence can be represented as a set of pairs (a_k, α_k) where a_k is an event and, α_k is its recursion depth. The distance of two execution sequences is a representation of their difference ([Vuong92, Vuong97, Zhu97]).

1.4 Test selection

The methodology of conformance testing ([ISO9646]) defines test selection as a selection process of test cases from an Abstract Test Suite based on the PICS and the PIXIT documents. The test laboratory selects the set of test cases the corresponding protocol behaviour of which are actually implemented in the IUT. This is usually an automatic selection and keeps the mandatory capabilities, as well as optional and conditional capabilities in view during the selection process ([Baum94]).

In this thesis, however, we will use test selection in a different context. In our terms test selection means selecting test cases from a given test suite in order to get a test set which is executable within limited resources. During the conformance testing process, the preparation for testing and the test operation is time intensive and expensive. For this reason, test laboratories usually have to omit some test cases and execute only certain ones ([Heijink94]). By this selection, the soundness of the test suite is kept, though its

error-detecting capability may decrease. This is an actual and very important practical problem and there is no existing theoretical method to solve, though "test selection should not be done at random, but a strategy should be applied, such that the resulting reduced test suite is valuable for conformance testing" ([Tretmans92]).

In this section we present three different approaches to test selection. They differ in the given test suite which is the basis of the selection. In the first, a test set is obtained through test generation; the second employs execution sequences; and the third, which is the subject of this thesis, uses the Abstract Test Suite as a basis of this selection. But first of all we introduce a very important notation: the *coverage*.

The coverage is a widely used, though not exactly defined, metric to measure the completeness of a set of test cases with respect to checking the conformance of a protocol implementation. Generally, the coverage of a test suite quantifies the percentage of the protocol behaviour "covered". In other words, the coverage is a quantification of the error-detecting capabilities of a test suite. A coverage measure can be used to compare test suites; high coverage expresses high quality tests ([FMCT]).

1.4.1 Selecting from a generated test set

In theory, the test generation algorithms produce optimal test suites, so there is no need to select test cases from the generated test set. In practice, however, the test generation algorithms may generate a very large number of test cases, hence test selection is required. In this case selection methods described below can be applied. Note that the practical importance of this approach is questionable, as the test generation methods are very rarely used in practice.

1.4.2 Selecting from the ATS

The methods described above rely on test generation or use execution sequences as in the Metric Based Test Selection method ([Vuong92, Vuong97, Zhu97]). The main drawback of these approaches is that formal descriptions (either FSM or execution sequences) of real-life protocols are most frequently lacking. We overcame this problem by using the formal ATS as a basis of selection.

Our selection method focuses on the ATS, which is the only standardized, available form in conformance testing. As we saw earlier, in using other test selection methods the formal descriptions of the protocol specifications are necessary, however, those are not

available in many cases. That is why our method selects TCs from the ATS, so the testers can execute them directly.

This approach is based on the practice of test selection used in test laboratories. We assume that we are given an ATS and we cannot generate any new test cases. In other words, we follow the practice of test laboratories as they can use only the test cases provided by the standards. Up to now, the selection of test cases from the ATS has been based on a subjective decision of the test laboratory. Our goal is to provide a theoretical background to this selection.

The question that arises in this situation is what test cases should be selected. An ideal choice would be a test set that checks as large part of the protocol specification as possible, that is, has as large coverage as possible, and needs as little resources as possible.

As the whole ATS is usually not complete, the requirements it tests may not cover the entire behaviour of the protocol. However, since the task is to select test cases of a given ATS, the coverage of a set of test cases can be viewed as its relative completeness with respect to the whole ATS. That is, coverage measures how much the requirements of the test suite are covered by the selected test cases. In other words, we can assume the coverage of the whole ATS to be 100%.

A practical test selection method should be efficient, in that it should make the test process faster by losing only a relatively small part of coverage. It is evident that we would like to save time during the testing process and get as high quality test suites as possible.

We would like to simplify the conformance testing process performed by test laboratories. The laboratories have to test several implementations of different vendors, so the test selection method has to be applicable to different types of real-life protocols.

The method has to be flexible, meaning that special preferences of the test laboratories and their knowledge of the specific protocol specification can be incorporated.

An important requirement of any kind of testing is that it has to be reproducible, that is, the selection can be repeated if it is needed and the same input data should result in the same selected test set, to ensure objective decisions.

We satisfy these requirements by introducing a mathematical model of the selection and applying optimization techniques.

A Further requirement is that the test selection process should necessitate as few human intervention as possible, so in other words, it has to be automated. We will describe this automation in the next chapter.

Chapter 2

Automation in test selection problem

In the previous chapter we outlined the task of conformance testing and presented the test selection problem. We listed the requirements an efficient test selection method has to fulfill: it has to be efficient and reproducible. Furthermore, an efficient test selection method has to be stand-alone, that is, it has to involve as few human intervention as possible. These requirements need the automation of the test selection process. On the other hand, as we are going to apply a mathematical model to the problem of test selection, we have to formalize it.

In this chapter we give a formal description of the basic notations of conformance testing presented previously and introduce a new notation: the subpurposes. They are mathematically well defined parts of a protocol specification which are automatically detectable in the ATS. Finally, we present how the data elements can be used in the test selection method.

2.1 Definition of subpurposes

Given the standardized test suite $TS = \{t_1, t_2, \ldots, t_n\}$ consisting of test cases $t_1, t_2, \ldots, t_n$. The set of corresponding test purposes $TP = \{p_1, p_2, \ldots, p_n\}$ is also given. As we explained in Section 1.1.3, there is a one-to-one connection between the test cases and test purposes: p_i corresponds to t_i $(i = 1, 2, \ldots, n)$.

Let $REQ = \{r_1, r_2, \ldots, r_m\}$ denote the set of conformance requirements. As defined in Section 1.1.3, test purposes are related to a set of conformance requirements, and a test case is written to check the requirements involved in the corresponding test purpose. The

following relation describes this connection between the test cases and the conformance requirements:

$$t_i \ \underline{check} \ r_j \stackrel{def}{\Leftrightarrow} t_i \text{ is able to check } r_j \tag{2.1}$$

Although test purposes describe conformance requirements, this description is prose and informal. To apply formal methods we need a formal representation of test purposes, so let us introduce the *abstract test purpose* P_{t_i} for each test case t_i as a set of conformance requirements the test case is able to check:

$$P_{t_i} \stackrel{def}{=} \{r \in REQ \mid t_i \, \underline{check} \, r\} \quad (i = 1, 2, \ldots, n) \, .$$

Note that in the ETSI terminology ([ETR266]) "formal test purpose" is used to denote a special format of test purposes. This format is a structured description, but it is not suitable for automated processing. Moreover, conformance requirements, which describe the basic features of the protocol behaviour that have to be checked to ensure the conformance of the protocol, are not given in an appropriate, automatically processable or even formal format.

Hence, neither the test purposes nor the conformance requirements are available in a format that is suitable for automated test selection. That is why we define subpurposes.

Definition 3. The *subpurposes* are automatically detectable approximations of the conformance requirements.

Figure 2.1 illustrates the relationship between conformance requirements, test purposes and subpurposes. Test purposes are made up of conformance requirements, while the requirements can be associated with a set of subpurposes.

Conformance requirements provide a subdivision of the protocol behaviour: every relevant aspect of the protocol specification is described with one or more requirements. Subpurposes provide another partition. As both the entire set of conformance requirements and the subpurposes cover the protocol behaviour, the building blocks of the original partition (i.e.: the conformance requirements) can be described by the building blocks of the second partition (i.e.: the subpurposes).

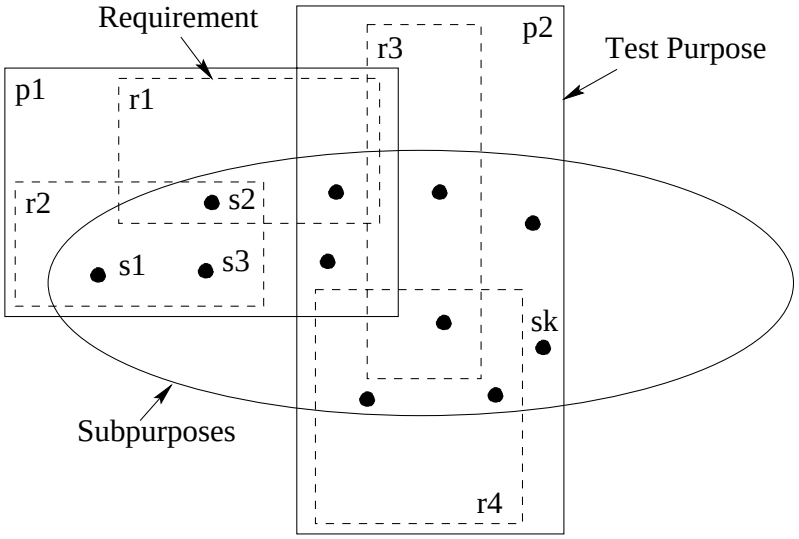

Figure 2.1: Requirements, test purposes and subpurposes

To formulate this relationship, let $SP = \{s_1, s_2, \ldots, s_k\}$ be the set of subpurposes and for each requirement we associate a subset of SP:

$$r_j \longmapsto SP_{r_j} \subseteq SP \text{ for } j = 1, 2, \ldots, m \tag{2.2}$$

We can extend the *check* relation to subpurposes by defining the set of subpurposes that are checked by test case t_i ($i = 1, 2, \ldots, n$):

$$\begin{gathered} t_i \longmapsto S_{t_i} \subseteq SP \text{ where } S_{t_i} \stackrel{def}{=} \{s \in SP \mid \exists r \in REQ,\, r \in P_{t_i},\, s \in SP_r\} \\ t_i \ \underline{check}\ s_j \stackrel{def}{\Leftrightarrow} s_j \in S_{t_i} \end{gathered} \tag{2.3}$$

So far we have described the theoretical way of introducing subpurposes and defining sets of S_{t_i}. In practice, the requirements and therefore the sets P_{t_i} are not given, so we have to define the sets S_{t_i} directly and skipping the association described in (2.2). We introduce this method in the next section.

As a summary, we classified the most important notions of test selection with respect to their properties (Table 2.1). Test cases and test purposes are available in standardized test documents, while subpurposes and requirements are not. (Note that in some rare cases, requirements are explicitly listed as in the Technical Basis for Regulation of ISDN primary rate access ([TBR4]). A detailed description of these protocol standards can be found in Section 4.1.1.)

The difference between requirements (given or not given) and subpurposes is that subpurposes are automatically detectable by definition.

Table 2.1: The classification of the most important notions of test selection

	Automatically detectable	Not automatically detectable
Given	test case	test purpose
Not given	subpurpose	requirement

2.2 Determination of subpurposes

In the previous section we introduced subpurposes as automatically detectable parts of the protocol behaviour which can approximate the conformance requirements. In this section we present a possible way to determine subpurposes.

The conformance requirements have four components ([Tarnay91]):

- Messages and their parameters
- Time periods
- States
- Protocol mechanisms

Furthermore, standard [ISO9646] deals with specification of the test suite structure, test purposes and the coverage of the conformance requirements of the relevant protocol specification. The standard advises the test suite specifier to design the test suite structure and test purposes focusing on PDUs sent to or received from the IUT, on the values of individual parameters, etc. Moreover, according to the standard ([ISO9646]), OSI protocol specifications define dynamic conformance requirements in terms of PDUs and the ASPs, that is, data elements.

On the other hand, messages and their parameters from the list above, in other words the data, play an increasingly important role in recent protocols. Take for example the modern WWW protocols (HTTP1.1), or the latest internet protocol (IPv6).

In summary, [ISO9646] informally suggests that a test suite specifier should use data elements when designing an abstract test suite. For these reasons, it is rational to utilize this approach also in our test selection method to determine the subpurposes, therefore let

make the data elements equivalent with the subpurposes. That is, we relate the abstract test purpose of a test case to the set of data elements sent or received in the test case. Of course, this way we get only an approximation of the real purpose of the test case but as experimental results will show, this approach can be justified.

On the other hand, data elements used in a test case can be detected automatically with the help of the TTCN machine processable format of an ATS.

To make it formal, let $DATA = \{d_1, d_2, \ldots, d_q\}$ be the data elements used by the protocol. To every test case t_i we assign the protocol data elements sent and received in t_i:

$$D_{t_i} \stackrel{def}{=} \{d \in DATA \mid d \text{ used in } t_i\}$$

Now, as said above, let us map the data elements onto subpurposes, namely $SP := DATA$, by identifying the set of subpurposes checked by a test case with the set of data elements used in it:

$$S_{t_i} := D_{t_i}, \quad (i = 1, 2, \ldots, n) \tag{2.4}$$

2.3 Abstract data levels

Data elements are not well-defined notions in conformance testing, for different reasons. First, because protocol specifications are different with respect to data content. There are protocol specifications where the data have much more importance than the behaviour control part, while in other protocol specifications there is more emphasis on the control. In this latter case, the variety of data is usually poor and its characterizing part lies "deeper", for example, in the parameters of the messages.

Secondly, different standardization institutes and test suite specifiers issue ATSs written in different ways. One prefers to use few, highly parameterized ASPs and PDUs, and the other employs more ASP and PDU constraints.

To get a generic view of data elements, let us now examine what is common in the data of every ATS. Evidently, there is always a hierarchy in data elements. There are data elements on higher or lower level. Thus, let us define the *abstract data levels* as a generic frame to describe this hierarchy. A simple possible set of abstract data levels is the following:

1. level of *data type*
2. level of *data instance*
3. level of *data parameter*

This model is the simplest possible one, as further levels, such as the level of the parameters' parameters can be included, depending on the structure of the data.

To determine the subpurposes, we have to decide which level is the most important in the ATS. The level which expresses most of the protocol behaviour will play the role of subpurposes. This decision is made intuitively, based on the protocol specification and the way the ATS was specified.

If the ATS is written in TTCN, the data levels are the following:

1. ASP and/or PDU type level
2. ASP and/or PDU constraint level
3. parameters of ASPs and/or PDUs: simple types, structured types, etc...

In 2000, ETSI standardized the new version of TTCN (TTCN version 3 [TTCN-3]). The TTCN-3 does not distinguish between ASP and PDU types, so it is necessary to change the above structure:

1. message type level
2. template level
3. parameters of template level

Naturally, the previously presented abstract data levels apply for the newer versions of TTCN, or any other testing languages.

Chapter 3

Mathematical model for test selection

3.1 Problem formulation

Let us recall that $TS = \{t_1, t_2, \ldots, t_n\}$ denotes the test suite and $SP = \{s_1, s_2, \ldots, s_k\}$ consists of the subpurposes related to it. We define a positive *cost* function $c : TS \to \mathbb{R}_+$ measuring the amount of resources the execution of a test case requires. The relative importance of the subpurposes with respect to the correct behaviour of the protocol is represented by the *weight* function $w : SP \to \mathbb{R}_+$. We will discuss in Section 4.2 how these functions can be determined.

In (2.3) we defined S_{t_i}, the subset of subpurposes test case t_i is able to check. From another point of view, a subset of test cases, T_{s_i} is associated with each subpurpose s_i $(i = 1, 2, \ldots, k)$ containing the test cases which are able to check the subpurpose:

$$T_{s_i} \stackrel{def}{=} \{t \in TS \mid t \,\underline{check}\, s_i\} \quad (i = 1, 2, \ldots, n)\,.$$

We will call these tests *related to the subpurpose.* That is, a test case is related to a subpurpose if it can give information about the subpurpose. For the sake of simpler notations, let b_i denote the number of test cases related to subpurpose s_i, namely let $b_i = |T_{s_i}|$ for each subpurpose.

Following the formal description presented above, we can define the **subpurpose-test case incidence matrix** A (Figure 3.1). Subpurposes $(s_1, s_2, \ldots, s_k)$ are placed in the

rows, test cases $(t_1, t_2, \ldots, t_n)$ are placed in the columns. Matrix A has the following entries:

$$A(s_i, t_j) = \begin{cases} 1 & \text{if } t_j \underline{check}\, s_i \\ 0 & \text{otherwise} \end{cases}$$

c =		c(t1)	c(t2)	. . .	c(tn)
w =		t1	t2		tn
w(s1)	s1	1	0	. . .	1
w(s2)	s2	1	1	. . .	0
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
w(sk)	sk	0	1	. . .	1

Figure 3.1: Subpurpose-test case incidence matrix

Let T be an arbitrary set of the test cases, $T \subseteq TS$. The cost of this test set is defined as the sum of the costs of the test cases belonging to T:

$$c(T) = \sum_{t \in T} c(t) \tag{3.1}$$

Let $cov(T)$ denote the coverage of test set T, that is, $cov(T)$ measures how large the part of the protocol specification tested by T is (below, in (3.4), we give a formal definition as well).

Two optimization problems can be defined according to our purposes and limitations.

Minimal cost problem: Given a lower bound (K) for the coverage. Find the set of test cases that satisfies this bound with minimal cost.

$$\begin{aligned} \min \quad & c(T) \\ \text{subject to} \quad & cov(T) \geq K \\ & T \subseteq TS \end{aligned} \tag{3.2}$$

Maximal coverage problem: Given an upper bound (L) for the cost. Find the subset of test cases the cost of which is not more than L and checks as large part of the

protocol specification (i.e.: has as big coverage) as possible.

$$\begin{aligned} \max \quad & cov(T) \\ \text{subject to} \quad & c(T) \leq L \\ & T \subseteq TS \end{aligned} \tag{3.3}$$

Subpurposes represent separate parts of the protocol behaviour and their importance is denoted by their weights. This implies that the coverage of a test set can be expressed as the weighted sum of the individual coverages of the subpurposes.

$$cov(T) = \sum_{s_i \in SP} w(s_i) \cdot purpcov(s_i, T) \tag{3.4}$$

where $purpcov(s_i, T)$ measures in what proportion T covers the basic conformance feature represented by subpurpose s_i. Naturally, $0 \leq purpcov(s_i, T) \leq 1$ for all $s_i \in SP$ and $T \subseteq TS$. It is assumed that $purpcov(s_i, T)$ is a function of the number of test cases that are related to s_i and belong to test set T:

$$purpcov(s_i, T) = f_i(|T_{s_i} \cap T|) \quad (i = 1, 2, \ldots, k)$$

for some $f_i : \{0, 1, \ldots, |T_{s_i}|\} \to [0, 1]$. The exact values of these functions depend on the methodology the construction of the test suite followed, but we can make some common assumptions which have to be true for any choice of f_i. Possible alternatives will be presented in the next section.

3.2 Coverage models

If test set T contains none of the test cases related to s_i, then executing T gives no information about the subpurpose, hence $f_i(0) = 0$. If T contains all the related test cases, then we have all the available information, so $f_i(b_i) = 1$. Between these two extreme values, let us suppose that adding one more related test case to T increases the coverage with a specific value e_i, and the subpurpose is supposed to be entirely covered if $|T_{r_i} \cap T|$ reaches D_i. With formulas:

$$f_i(m) = \begin{cases} me_i & \text{if } m < \min\{\lceil 1/e_i \rceil, D_i, b_i\} \\ 1 & \text{otherwise} \end{cases}$$

In this way D_i and e_i fully determine f_i. Below we list four special cases which model the possible real-life situations the best. We call them *coverage models* as they determine the coverage of a test set. The coverage models are graphically demonstrated in Figure 3.2.

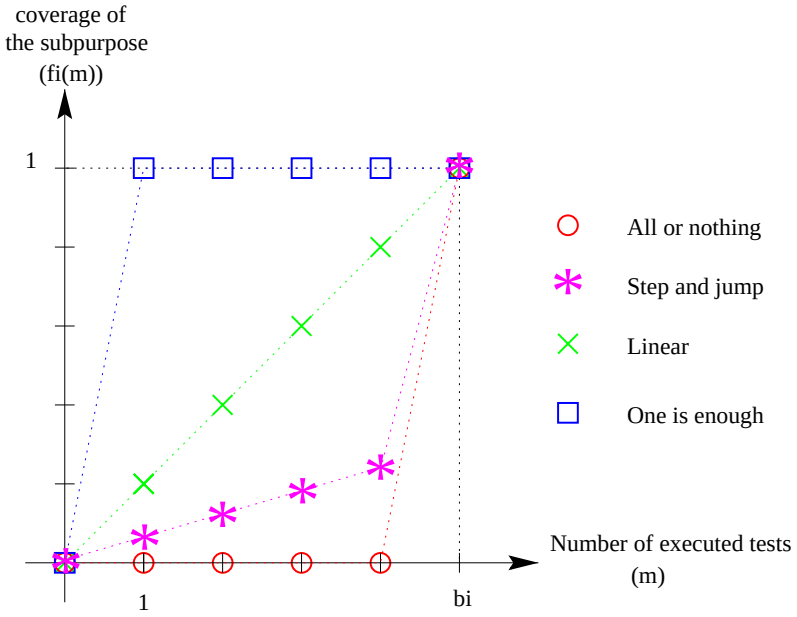

Figure 3.2: Coverage models

1. **All or Nothing model**: $D_i = b_i$ and $e_i = 0$, so $f_i(0) = f_i(1) = \ldots = f_i(b_i - 1) = 0$ and $f_i(b_i) = 1$. This model implies that T cannot test anything of s_i unless it covers the subpurpose entirely. In other words, executing $b_i - 1$ test cases from T_{s_i} has the same result as executing none of them.

2. **Step and Jump model**: $D_i = b_i$ and $e_i = 1/|TS|$, so

$$f_i(m) = \begin{cases} m/\mid TS \mid & \text{if } m < b_i \\ 1 & \text{if } m = b_i \end{cases}$$

In this model we assume that adding one more related test case to T increases the coverage with a relatively small amount until all of them belong to T, when the coverage is 1. If only a few test cases are related to s_i, then executing all but one has a small contribution to the coverage, while if $|T_{s_i}|$ is big, then we can obtain high coverage by executing only a part of the related test cases.

3. **Linear model**: $D_i = b_i$ and $e_i = 1/b_i$, so $f_i(m) = m/b_i$ for $m = 0, 1, \ldots, b_i$. This model means that the coverage is in direct proportion to the number of related test cases belonging to T.

4. **One is Enough model**: $D_i = 1$ and $e_i = 1$, so $f_i(0) = 0$ and $f_i(1) = \ldots = f_i(b_i) = 1$. That is, we consider the whole subpurpose to be entirely tested if at least one of the related test cases is in T.

Chapter 4

Test selection process

In this section we show how the theory can be turned into practice. We give a possible approach to the test selection process using the previously presented notions and mathematical apparatus.

Figure 4.1 illustrates the main phases of the test selection process. The main steps are:

1. creating the subpurpose-test case incidence matrix manually or automatically
2. determining the cost and weight functions (c and w), deciding which optimization problem (minimal cost or maximal coverage) and coverage model (All or Nothing, Step and Jump, Linear or One is Enough) will be used, inputting the cost or coverage bound (L or K), and choosing a suitable optimization algorithm
3. loading the selected test cases into the tester

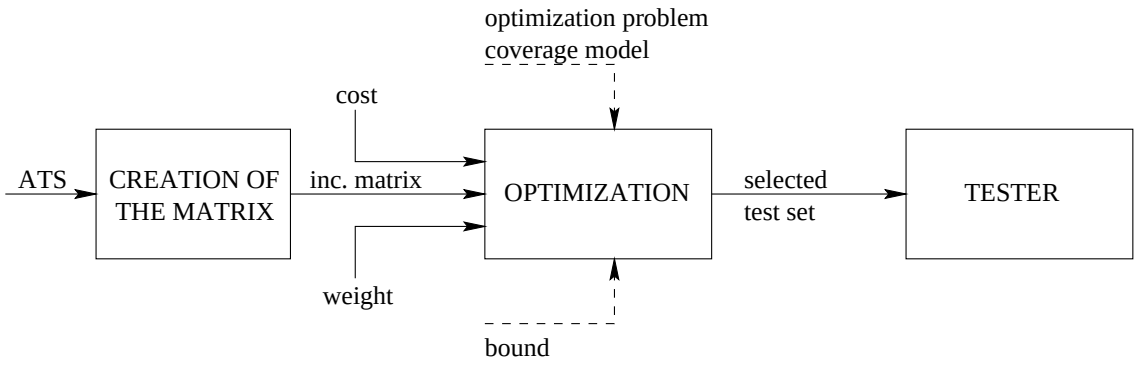

Figure 4.1: The process of selection

For a given ATS of a protocol, the first step (i.e.: the creation of the incidence matrix) has to be done only once. This step can be the most time consuming of the test selection

process and it is where the most human interface is necessary. (If done with the manual matrix filling method, described later). The fact that the same incidence matrix can be used for subsequent selections means that the efficiency of test selection can be strongly improved once the incidence matrix is prepared

Furthermore, if the incidence matrix is available, we can try to carry out the optimization for different cost, weight functions, optimization problems or different coverage models and bounds in order to find the best test set that meet our aims.

4.1 Creation of the subpurpose-test case incidence matrix

In the test selection process, the most crucial step is the creation of the subpurpose-test case incidence matrix. Only the incidence matrix is able to hold the information of the protocol specification, that is why it is so important to fill this matrix precisely. On the other hand, this matrix filling needs to be done only once for a protocol specification, and it is reusable for any optimization method later as many times as needed.

4.1.1 Manual matrix filling

There are protocols for which the conformance requirements are given in the ATS standard and they are accompanied by the test cases that can be used to check them. In this case we do not need to deal with deriving subpurposes because we can easily construct a matrix that describes the connection between the requirements and the test cases. As an example, we will perform the test selection for the ISDN primary rate DSS1 Layer 2 protocol ([TBR4]) in Section 8.1. The Technical Basis for Regulation (TBR) document explicitly lists the requirements and the related test cases, as Figure 4.2 shows.

In other cases, when the requirements are not given explicitly, they have to be determined or approximated by subpurposes. If the ATS is not given in a machine processable form, (i.e.: in TTCN.MP) the incidence matrix can be filled only manually. In this case we cannot speak about subpurposes as they have to be automatically detectable by definition. What we do instead is finding the conformance requirements implicitly given in the protocol description and the test cases and fill the requirement-test case incidence matrix by examining which test case checks which requirement. This is a difficult task and requires deep knowledge of the chosen protocol and test description, as well as considerable time for real-life protocols. On the upside, the matrix filled by an expert is the one which is the closest to the theoretical conformance requirement-test case incidence matrix.

10.7.5.2 Expiry of timer T200 during "peer receiver busy"

Requirement: If timer T200 expires, the data link layer entity shall:

- if it is not yet in a timer recovery condition, enter the timer recovery condition and reset the retransmission count variable; or
- if it is already in a timer recovery condition, add one to its retransmission count variable.

The data link layer entity shall then:

a) if the value of the retransmission count variable is less than N200
   - transmit an appropriate supervisory command (see 10.7.5.5) with a P bit set to 1;
   - restart timer T200; and

b) if the value of the retransmission count variable is equal to N200, initiate a re-establishment procedure as defined in subclause 10.8, and indicate this by means of the MDL-ERROR-INDICATION primitive to the connection management entity.

Test: The test shall be conducted according to annex C test cases TC27411 and TC27417.

Figure 4.2: An example for a requirement and the related test cases in ETSI TBR4

Telecommunication companies (e.g. Ericsson) use a test methodology called *function test*, which uses neither the conformance testing methodology nor TTCN. For this reason it is impossible to create incidence matrix automatically based on TTCN.MP and the manual matrix filling has to be used. In Section 8.2 we will expand our test selection methodology on function tests using the BGC Attendant ISDN-E Connection ([BGC]) test description.

4.1.2 Automatic matrix filling

When a machine processable form of the ATS (usually TTCN.MP) is given, then the cumbersome manual matrix filling can be avoided by using subpurposes. Subpurposes are defined as automatically detectable parts of the protocol's behaviour (see Section 2.1), so automatic methods can be used to fill the subpurpose-test case incidence matrix.

A good subpurpose definition result in closer approximation of the theoretical requirement-test case incidence matrix, so one has to pay due attention to this step. On the other hand, the determination of the subpurposes needs only a good general view of the structure of the ATS and the behaviour of the protocol. It does not require detailed examination and comparison of test cases as in the case of the manual matrix filling where the conformance requirements and their relation to the test cases have to be fully detected.

We have to decide first what abstraction level of the protocol is going to play the role of subpurposes, in other words, following Section 2.2, we have to choose an abstract data level (see Section 2.3) which will represent the subpurposes. Although PDUs are

the most natural candidates for subpurposes, sometimes it is worth stepping one layer up or down, and choosing the PDU types or the parameters of the PDUs respectively to be subpurposes, if it results in a more efficient selection. The most appropriate choice of subpurposes depends on the structure of the ATS.

We implemented a PERL program that is able to find the PDUs used in the test cases and fill the incidence matrix independently of the size of the ATS. This program can also detect PDUs that are used in the test steps or defaults appearing in the test case. So the PDUs related to a test case are sent either in its main body or in a test step or default used in the test case. By this, the dependencies of test cases are handled. For example, when a test case appears as the preamble of another test case, the incidence matrix will reflect this fact, consequently this dependency is taken into account in the optimization. The program also deletes the all zero rows if there are any (i.e.: eliminate the PDUs not used in the ATS), and merge equal rows, because in this case the PDUs related to these rows can be together viewed as one subpurpose.

4.2 Creation of cost and weight vectors

Determining the costs of the tests and the weights of the subpurposes is up to the test laboratories' individual preferences, but we expect that choosing them to be the same for each test case and PDU respectively is the most natural solution. As for the coverage, this is because at first sight there is no theoretical basis to distinguish between the PDUs. The costs can be chosen to be the same because the most time consuming task during the conformance assessment process is preparation and parameterization, the time of which is in direct proportion to the number of the executed tests. Most often the execution time is negligible compared to the time of preparation and parameterization. Therefore, in most cases all-one vectors can be used as cost and weight vectors. In our experiments we also used randomly generated vectors, cost that reflected the execution time of the tests, as well as costs and weights based on the estimation of experts.

Chapter 5

Mathematical solution methods

Chapter 3 presented the two optimization problems that can arise in test selection. In this chapter we discuss the possible methods to solve the Mathematical Programming problems (3.2) and (3.3). First, we have to determine the complexity of the problems.

The complexity of a problem can be expressed as the time required by a solution algorithm in the worst case. If a problem falls into the class NP-hard, then there is not known (and cannot be expected) an algorithm that will always be able to solve the problem in time which is a polinomial function of the input size. That is to say, if an optimization problem is NP-hard, then we cannot expect to find an optimal solution for any input in a reasonable amount of time.

We show that the test selection problem is NP-hard. The minimal cost problem in the One is Enough model (i.e.: if all requirements follow the One is Enough coverage model) for $K = \sum_{r \in REQ} w(r)$ is equivalent to the Set Covering Problem, which has been proven to be NP-hard ([Garey79]). This implies that the general minimal cost problem is NP-hard. Furthermore, if $|TS| = |REQ|$ and $T_{r_i} = \{t_i\}$ for all i, then $cov(T) = \sum_{\{i:t_i \in T\}} w(r_i)$ in each model. Thus, this special case of the test selection problem is equivalent to the knapsack problem ([Garey79]). Consequently, the test selection problem (both the minimal cost and maximal coverage) for general cost and weight functions is NP-hard, independently of the coverage model.

Although the test selection problem is NP-hard, this does not mean that it is surely impossible to find an optimal solution for a given input. And in cases when it is really not possible, powerful heuristic methods (e.g.: Tabu Search, Genetic Algorithms and Simulated Annealing) can be applied to find good quality suboptimal solutions.

There is no general method to solve Mathematical Programming problems to optimum. However, if it can be transformed into an *Integer Linear Programming* (ILP) problem, then well established algorithms can be employed. The most widely used, the *Branch and Bound* (B&B) method ([Nemhauser98, Wolsey98]), is implemented in several commercial softwares.

In this chapter we show that the test selection problem can be formulated as an ILP problem. This opens the door for applying efficient B&B softwares. On the other hand, we present heuristic approaches, too. They include simple greedy algorithms as well as more complex meta-heuristics such as Reactive Tabu Search (RTS), Genetic Algorithms (GA), and Simulated Annealing (SA).

Here, and throughout the rest of the thesis we will use the following reformulation of (3.2) and (3.3). Let $\mathbf{a_i}$ be the n-dimensional characteristic (row) vector of T_{s_i} for $(i = 1, \ldots, k)$, thus $\mathbf{a_i}(j) = 1$ if t_j is related to s_i. In other words, $\mathbf{a_i}$ is the i^{th} row of the subpurpose-test case incidence matrix. Let $\mathbf{c} = (c_1, \ldots, c_n)$ be the vector consisting of the costs of the test cases. Similarly, let $\mathbf{w} = (w_1, \ldots, w_k)$ denote the weight vector of the subpurposes. A test set $T \subset TS$ can be represented by its characteristic vector $\mathbf{x}_T$. Using these notations the cost and coverage of T are $c(T) = \mathbf{c}\mathbf{x}_T$ and $cov(T) = \sum_{i=1}^{k} w_i f_i(\mathbf{a_i}\mathbf{x}_T)$ according to (3.1) and (3.4). Hence the minimal cost problem (3.2) can be written as

$$\begin{array}{rl} \min & \mathbf{c}\mathbf{x} \\ \text{subject to} & \sum_{i=1}^{k} w_i f_i(\mathbf{a_i}\mathbf{x}) \geq K \\ & \mathbf{x} \in \{0,1\}^n \end{array} \tag{5.1}$$

and the maximal coverage problem (3.3) is

$$\begin{array}{rl} \max & \sum_{i=1}^{k} w_i f_i(\mathbf{a_i}\mathbf{x}) \\ \text{subject to} & \mathbf{c}\mathbf{x} \leq L \\ & \mathbf{x} \in \{0,1\}^n \end{array} \tag{5.2}$$

5.1 ILP formulation

We will show how the minimal cost and maximal coverage problem can be transformed to an Integer Linear Programming problem.[1] Let us see how formula (5.1) and (5.2) look like in the four coverage models introduced in Section 3.2.

[1] This ILP formulation was worked out by my colleague, Balázs Kotnyek. Its results will be compared with the later discussed heuristic results, so it is important to understand this formulation.

1. **Linear model:**

$$\sum_{i=1}^{k} w_i f_i(\mathbf{a_i x}) = \sum_{i=1}^{k} w_i \frac{\mathbf{a_i x}}{b_i} = \left(\sum_{i=1}^{k} \frac{w_i \mathbf{a_i}}{b_i} \right) \mathbf{x} \tag{5.3}$$

Thus, (5.1) turns into a binary minimization problem with a single linear constraint:

$$\begin{array}{rl} \min & \mathbf{cx} \\ \text{subject to} & \left(\sum_{i=1}^{k} \frac{w_i \mathbf{a_i}}{b_i} \right) \mathbf{x} \geq K \\ & \mathbf{x} \in \{0,1\}^n \end{array} \tag{5.4}$$

and (5.2) turns into a binary maximization problem:

$$\begin{array}{rl} \max & \left(\sum_{i=1}^{k} \frac{w_i \mathbf{a_i}}{b_i} \right) \mathbf{x} \\ \text{subject to} & \mathbf{cx} \leq L \\ & \mathbf{x} \in \{0,1\}^n \end{array} \tag{5.5}$$

Therefore, as mentioned earlier, (5.1) and (5.2) can be transformed into a knapsack problem. The existing efficient algorithms solving knapsack problems (see e.g.: [Nemhauser98, Wolsey98]) can be applied here.

2. **All or Nothing model:** Let us define a new variable vector $\mathbf{z} = (z_1, \ldots, z_k)$ in the following manner:

$$z_i = \begin{cases} 1 & \text{if } \mathbf{a_i x} = b_i \\ 0 & \text{if } \mathbf{a_i x} < b_i \end{cases} \tag{5.6}$$

In other words, $z_i = 1$ if s_i is tested by the test set represented by $\mathbf{x}$. Using this vector, problem (5.1) can be written as

$$\begin{array}{rlll} \min & \mathbf{cx} & & \\ \text{subject to} & \mathbf{a_i x} \geq & b_i z_i & i = 1, \ldots, k \\ & \mathbf{wz} \geq & K & \\ & \mathbf{x} \in & \{0,1\}^n & \\ & \mathbf{z} \in & \{0,1\}^k & \end{array} \tag{5.7}$$

and problem (5.2) as

$$\begin{array}{rrcll} \max & \mathbf{wz} & & & \\ \text{subject to} & \mathbf{a_i x} & \geq & b_i z_i & i = 1, \ldots, k \\ & \mathbf{cx} & \leq & L & \\ & \mathbf{x} & \in & \{0,1\}^n & \\ & \mathbf{z} & \in & \{0,1\}^k & \end{array} \tag{5.8}$$

3. **One is Enough model:** This model can be handled similarly to the previous one. Let $\mathbf{z}$ be the following vector:

$$z_i = \begin{cases} 0 & \text{if } \mathbf{a_i x} = 0 \\ 1 & \text{if } \mathbf{a_i x} \geq 1 \end{cases}$$

In this case (5.1) can be transformed into the following problem:

$$\begin{array}{rrcl} \min & \mathbf{cx} & & \\ \text{subject to} & A\mathbf{x} & \geq & \mathbf{z} \\ & \mathbf{wz} & \geq & K \\ & \mathbf{x} & \in & \{0,1\}^n \\ & \mathbf{z} & \in & \{0,1\}^k \end{array} \tag{5.9}$$

and in the case (5.2):

$$\begin{array}{rrcl} \max & \mathbf{wz} & & \\ \text{subject to} & A\mathbf{x} & \geq & \mathbf{z} \\ & \mathbf{cx} & \leq & L \\ & \mathbf{x} & \in & \{0,1\}^n \\ & \mathbf{z} & \in & \{0,1\}^k \end{array} \tag{5.10}$$

4. **Step and Jump model:** In this model $f_i(\mathbf{a_i x}) = \frac{1}{n}\mathbf{a_i x} + z_i\left(1 - \frac{b_i}{n}\right)$ with z_i defined in (5.6) and $n = |TS|$. Thus (5.1) is equivalent to

$$\begin{array}{rrcll} \min & \mathbf{cx} & & & \\ \text{subject to} & b_i z_i & \leq & \mathbf{a_i x} & i = 1, \ldots, k \\ & K & \leq & \frac{1}{n}(\mathbf{w}A)\mathbf{x} + \sum_{i=1}^{k} w_i\left(1 - \frac{b_i}{n}\right) z_i & \\ & \mathbf{x} & \in & \{0,1\}^n & \\ & \mathbf{z} & \in & \{0,1\}^k & \end{array} \tag{5.11}$$

and (5.2) is equivalent to:

$$\begin{array}{rrcll} \max & & & \left(\frac{1}{n}(\mathbf{w}A)\mathbf{x} + \sum_{i=1}^{k} w_i\left(1 - \frac{b_i}{n}\right) z_i\right) & \\ \text{subject to} & b_i z_i & \leq & \mathbf{a_i x} & i = 1, \ldots, k \\ & L & \geq & \mathbf{cx} & \\ & \mathbf{x} & \in & \{0,1\}^n & \\ & \mathbf{z} & \in & \{0,1\}^k & \end{array} \tag{5.12}$$

Given the ILP formulation, we experimented with efficient implementations of B&B available in commercial MIP solvers. Their performance was just like one would expect when dealing with NP-hard problems: sometimes they could find the optimal solution very fast, in other cases they found it slowly or found only a feasible integer solution within the computational limits. In a few cases they could not give us any feasible integer solution.

5.2 Heuristics

As the test selection problem is NP-hard, no algorithm exists that aims at an exact optimum in a reasonable time limit. However, the B&B methods that uses the ILP formulation described in the previous section are still applicable. The advantage of these methods is that if they stop within the time available, the solution they provide is surely optimal. Their drawbacks is that if they do not find an optimal solution within the time limit, and the algorithm is pruned, the best solution they can offer may be very far from the optimum. To avoid this unpleasant situation several heuristic algorithms have been developed (see e.g.: [Aarts97]). These methods cannot guarantee exact optimal solutions, but they are usually capable of giving a good approximation.

There is another, conceptual problem with seeking an exact solution. The methods used for determining the inputs (e.g.: the cost and weight vector or the subpurpose-test case incidence matrix) of the test selection problem, by their nature, are based on estimations and 'soft' considerations regarding the protocol (as discussed in Chapter 4). Hence, an exact optimal selection is only optimal up to the level of these approximations. This implies that the difference between the potentially inaccurate solution of a heuristic method and the exact optimum is only quantitative, not qualitative. Hence, a small quantitative difference is absolutely acceptable when considering the practice of testing. That is, if a solution, which is close enough to the best possible one can be obtained in a reasonable time, then any tester will opt for the inaccurate but faster method. This is quite sensible because the purpose of test selection is to save time during testing.

A further practical advantage of the heuristic methods is that they are simpler than the efficient B&B algorithms, so it is easier to introduce them into the testing process. This is especially true for the commercial implementations of B&B, the cost of which may also deter possible users.

Let us see the heuristic algorithms. An algorithm that returns near-optimal solutions is generally called a heuristic algorithm. A possible definition according to [Chu97] is:

> "A heuristic algorithm is a method that is designed to provide 'good' solutions but cannot guarantee optimality. It is generally intended for solving approximately and efficiently, large and/or difficult optimization problems for which optimal solutions cannot be found in a reasonable amount of computing time or within the available amount of computer memory by using any of the existing exact methods."

The best known heuristic methods can be categorized as follows:

- **Greedy heuristics:** Greedy algorithms seek immediate gain at each step and a feasible solution is usually found at the end of the procedure. Greedy heuristics are often regarded as computationally the most efficient of all heuristic approaches. However, the drawback of this approach it that the solution it obtains can be quite far from the true optimum. Greedy algorithms will be examined more closely in Section 5.2.1.

- **Neighbourhood search methods:** A neighbourhood search or local search heuristic begins with a feasible solution and successively improves it by a sequence of

exchanges or mergers in a local search. The search proceeds by moving from one feasible solution to one of its *neighbours* while improving the objective function and maintaining feasibility. If no new improved solution is found, a local optimum is obtained. Many neighbourhood search methods combine a greedy heuristic to obtain an initial starting solution with a local search in finding local optimum. In this thesis we will discuss in depth these kinds of algorithms, namely Tabu Search in Section 5.2.2, Genetic Algorithms in Section 5.2.3 and Simulated Annealing in Section 5.2.4. A good summary of neighbourhood algorithms can be found in [Pirlot96] and [Frank97].

In the following sections, we give the descriptions of the heuristic methods and algorithms as we applied them to the minimal cost problem (3.2). The maximal coverage problem can be handled very similarly.

5.2.1 Greedy algorithms

Test laboratories usually intuitively use one of the following two simple greedy algorithms (Algorithm 1 ADD and Algorithm 2 DROP) for selecting the executed test set T if a lower bound K for its coverage is given. These simple algorithms can model the human or subjective test selection made by the testers in test laboratories, so they give a possibility for a comparative analysis with the later discussed, more complex heuristic algorithms.

Algorithm 1 ADD

Step 1: Let $T = \emptyset$.
Step 2: Find a test case t_i in $TS \setminus T$ with minimal cost. If there are more, then choose the one with the lowest index. Let $T = T \cup \{t_i\}$.
Step 3: If $cov(T) \geq K$, then STOP, otherwise go to *Step 2*.

Algorithm 2 DROP

Step 1: Let $T = TS$ and $U = TS$.
Step 2: If $U = \emptyset$, then STOP. Otherwise find a test case t_i in U with maximal cost. If there are more, then choose the one with the lowest index. Let $U = U \setminus \{t_i\}$.
Step 3: If $cov(T \setminus \{t_i\}) \geq K$ then $T = T \setminus \{t_i\}$ and go to *Step 2*.

We propose two more sophisticated algorithms (Algorithm 3 ADD-DELTA and Algorithm 4 DROP-DELTA) for the minimal cost problem. The greatest problem with the two previous algorithms is that in choosing a test case to add or drop, they do not take into consideration the change in the coverage it may imply. To overcome this problem, we

calculate the marginal coverage of each test case, that is, the change in the coverage as a consequence of the change in T. We compare it with the change in cost, and choose the test case to add to or drop from T that proves to be the best.

Algorithm 3 ADD-DELTA

Step 1: Let $T = \emptyset$.
Step 2: For each $t_i \in TS \setminus T$ calculate the increase in coverage and cost if it is added to T: $\Delta cov(i) = cov(T \cup \{t_i\}) - cov(T)$ and $\Delta cost(i) = c(T \cup \{t_i\}) - c(T)$.
Step 3: Find a test case t_i in $TS \setminus T$ for which $\frac{\Delta cost}{\Delta cov}$ is minimal. If there are more, then choose the one with the lowest index. Let $T = T \cup \{t_i\}$.
Step 4: If $cov(T) \geq K$, then STOP, otherwise go to *Step 2*.

Algorithm 4 DROP-DELTA

Step 1: Let $T = TS$ and $U = TS$.
Step 2: If $U = \emptyset$, then STOP. Otherwise, for each $t_i \in U$ calculate the decrease in coverage and cost if it is excluded from T: $\Delta cov(i) = cov(T) - cov(T \setminus \{t_i\})$ and $\Delta cost(i) = c(T) - c(T \setminus \{t_i\})$.
Step 3: Find a test case t_i in U for which $\frac{\Delta cost}{\Delta cov}$ is maximal. If there are more, then choose the one with the lowest index. Let $U = U \setminus \{t_i\}$.
Step 4: If $cov(T \setminus \{t_i\}) \geq K$, then $T = T \setminus \{t_i\}$ and go to *Step 2*.

We showed earlier that the minimal cost problem in the Linear model is a knapsack problem. Let $min\{\mathbf{cx} \mid \mathbf{vx} \geq K,\ \mathbf{x} \in \{0,1\}^n\}$ be this knapsack problem (so $\mathbf{v} = \sum_{i=1}^{k} \frac{w_i \mathbf{a}_i}{b_i}$ according to (5.3)). Let us suppose that $\frac{c_1}{v_1} \leq \frac{c_2}{v_2} \leq \ldots \leq \frac{c_n}{v_n}$ and l is the index for which $\sum_{i=1}^{l-1} v_i < K \leq \sum_{i=1}^{l} v_i$. Then

$$x_i = \begin{cases} 1 & \text{for } i = 1, \ldots, l-1 \\ K - \sum_{i=1}^{l-1} v_i & \text{for } i = l \\ 0 & \text{for } i = l+1, \ldots, n \end{cases}$$

is the optimal fractional solution. Thus changing x_l to 1 gives us a feasible integer solution of the original knapsack problem. In the Linear model $\Delta cov(i) = v_i$ and $\Delta cost(i) = c_i$ for every i and every T, so this integer solution is the same as what we get by applying Algorithm ADD-DELTA or Algorithm DROP-DELTA.

A powerful mixture of Algorithm ADD-DELTA and DROP-DELTA is realized in Algorithm 5 GREEDY, which will appear as a basic in the further heuristic algorithms.

Algorithm 5 GREEDY

Input: Initial test set T
Step 1. Adding new test cases:
Until $cov(T) \geq K$, do:
    For each $t_i \in TS \setminus T$:
        $\Delta cov(i) := cov(T \cup \{t_i\}) - cov(T)$
        $\Delta cost(i) := c(T \cup \{t_i\}) - c(T)$.
    Find a test case t_i in $TS \setminus T$ for which $\frac{\Delta cost}{\Delta cov}$ is minimal.
    If there are more, then choose the one with the lowest index.
    Let $T = T \cup \{t_i\}$
Step 2. Dropping redundant tests:
Let $U = T$
While $U \neq \emptyset$ do
    Find a test case t_i in U with maximal cost.
    If there are more, then choose the one with the lowest index.
    Let $U = U \setminus \{t_i\}$
    If $cov(T \setminus \{t_i\}) \geq K$ then $T := T \setminus \{t_i\}$

5.2.2 Tabu Search

Tabu Search ([Glover86, Glover89, Glover90]) is a local search technique, which uses adaptive memory to guide the search and to avoid local minima. A comprehensive description of Tabu Search along with references can be found in [Aarts97]. The basic notions of Tabu Search can be outlined as follows.

Let X be the set of feasible solutions and $\mathbf{x}_n$ be the current solution in iteration n. Its *neighbourhood* is a subset $V(\mathbf{x}_n)$ of X containing the solutions that can be obtained by a *move* from $\mathbf{x}_n$. The algorithm moves to the best possible solution in $V(\mathbf{x}_n)$, provided that it is not in the tabu list. A *tabu list* is used to prevent cycling by forbidding the search from moving to a certain set of solutions encountered recently. Usually, it contains special characteristics of the moves in the last iterations. An extra chance is given for good solutions inappropriately ruled out by too strict tabu lists: if a possible next solution is better than any solution met before, then we accept it even if it is in the tabu list (*aspiration criterion*).

The simplest Tabu Search algorithms require only *short-term memory* by keeping in memory only those visited solutions that are in the tabu list. In more sophisticated versions, the *long-term memory* is used, which enables us to use adaptive techniques. These techniques are usually applied to the tabu list, allowing its size to vary according to the search trajectory, which is stored entirely in the memory. A successful implementation of the adaptive tabu list size is given by Battiti and Tecchiolli in [Battiti94], where their

Reactive Tabu Search (RTS) algorithm is presented together with an application to the Quadratic Assignment Problem. In what follows, we summarize the basic features of RTS, as published in [Battiti94] and [Battiti96]. The solutions encountered during the search are kept in the memory. If a repetition of a previously visited solution occurs (if a cycle is detected), then the size of the tabu list is increased by a multiplicative factor. To avoid 'chaotic trapping', a second diversification mechanism is added. This is applied if too many solutions are repeated too many times. In this case an *escape* function is called which executes random moves to get the algorithm out of the trap. The size of the tabu list is decreased if more iterations passed since the last size change than the moving average of the cycles found recently, or if all the neighbours of a solution are tabu.

To handle the memory structure efficiently, hashing techniques [Knuth73] are applied. A standard version of it uses a *bucket hashing table*. In this case, the solution is represented as a binary number bn_x and the hash function is simply: $bn_x \mod pr$, where pr is a huge prime number denoting the size of the hash table. This way, the hash value of the solution is obtained, preferably different values for different solutions. If, however, a collision occurs, (i.e.: different solutions get the same hash value), a list of these solutions is linked to the value.

We applied RTS to the minimal cost test selection problem. According to formulation (5.1), the set of the feasible solutions X is a subset of $\{0,1\}^n$ containing the test sets the coverage of which is bigger than the given lower bound. We define $V(\mathbf{x}_n)$ as the set of 0-1 vectors in X which can be reached by flipping at one vector position in $\mathbf{x}_n$. That is, a move is equivalent to changing a bit in $\mathbf{x}_n$ from 0 to 1 or from 1 to 0. The tabu list contains the positions where the changes took place. We implemented the functions of RTS presented in [Battiti94]. Namely, our algorithm uses the *check_for_repetitions* function, which takes the current solution and a selected move as input and can result in either executing the move and changing the tabu list or calling the *escape* function if conditions are fulfilled. The selection of the move to be executed is done with function *choose_best_move*. Functions *check_for_repetitions* and *escape* are identical to the procedures presented in [Battiti94], while we substantially modified *choose_best_move* to incorporate a new technique, described later. Algorithm 6 gives the skeleton of RTS.

The initial solution is generated simply by setting the n-dimensional all-one vector. This proved to be more efficient than using the suboptimal solution generated with the greedy method of Algorithm 5, probably because in this latter case the RTS has difficulties in finding a neighbour of the initial solution.

Algorithm 6 REACTIVE TABU SEARCH (RTS)

Initialize the hash memory.
Generate an initial solution.
Iterative step:
    choose_ best_ move
    if all the neighbours are tabu, then decrease the tabu list size
    check_ for_ repetitions
    if there is no need to escape, then
        execute the move
        put the new solution into the hash table
    else
        escape
Stop, if a termination criterion is fulfilled.

The main difference between Battiti and Tecchiolli's RTS algorithm and our one is in function *choose_ best_ move*. Its task is to evaluate all the possible moves and find the best among those which are not tabu, or satisfy the aspiration criterion (see Algorithm 7). This involves checking the feasibility of every possible vector obtained from the current solution $\mathbf{x}$ by flipping one bit. Hence *choose_ best_ move* is the most critical part of RTS regarding execution time. To make it faster, we applied an *adaptive neighbourhood sampling* technique in combination with stochastic tie breaking. The rationale behind this technique is the following. It can easily happen that several different neighbours have the same best value, that is, any of these neighbours can be the next solution. In this case, it may be more efficient to evaluate only a subset of all the possible neighbours, as this subset will probably contain at least one of the best neighbours. The size of the evaluated subset should depend on the number of alternative best neighbours: the more alternatives we have had in the recent iterations, the smaller subset can be enough.

The subset of the evaluated neighbours is selected evenly by employing variable *step*, which determines the average distance between two selected positions in x. (See Algorithm 7.) If $step = 1$ (which is its default value), then every possible neighbour will be evaluated. If $step = 2$, then every second neighbour will be evaluated on average, and so on. The value of *step* varies adaptively. If the average number of best neighbours (avg_best_neigh) is greater than the required ratio $\frac{n}{nb}$ of the evaluated neighbours, then *step* is increased, otherwise it is decreased. We used the following functions $step := (\sqrt{step} + 0.1)^2$ and $step := (\sqrt{step} - 0.1)^2$. Global parameter nb determines the proportion of moves we wish to check. If nb is small, then we require a precise neighbourhood sampling, (i.e.: more neighbours are evaluated and possibly better move is found), but the algorithm takes more

Algorithm 7 FUNCTION *choose_best_move*

Initialization:
x denotes the current solution.
c^* is a global variable storing the best (minimum) cost found ever.
Reset *neighbour_cost*.

$i := 1$
While $i \leq n$ do:
Let y be a new neighbour of x obtained by flipping the i^{th} bit of x.
if y is feasible and (the move $x \rightarrow y$ is not tabu, or it is tabu but $cy \leq c^*$), then
$neighbour\_cost[i] := cy$
$i := \lfloor i + rnd \cdot step \rceil$
where rnd is a random number between 0.5 and 1.5
and $\lfloor x \rceil$ denotes the closest integer to x.

Find the minimum values of *neighbour_cost*
and count the number of the minimums (*min_count*).
$avg\_best\_neigh := 0.9 \cdot avg\_best\_neigh + 0.1 \cdot min\_count$
if $\left(avg\_best\_neigh > \frac{n}{nb}\right)$, then
increase *step*
else
decrease *step*.
Let j be a random integer number between 1 and *min_count*
and return with the position in x of the j^{th} minimum value.

time. If nb is large, then a more relaxed approach is applied: the algorithm will possibly not move to the best neighbour, but the execution time is reduced. We will discuss the effect of setting different values for nb in Section 6.2.

We terminated the algorithm after a fixed number (60 000) of iterations, or if there has been no improvement on c^* in the last 20 000 iterations.

5.2.3 Genetic Algorithms

Genetic Algorithms ([Holland75]) are search techniques developed based on natural selection. They operate on a set of feasible solutions called *population*. In each iteration a new population is generated from the existing one by applying genetic operators, the two most important of which are *crossover* and *mutation*. The values of the solutions, called *fitness*, are evaluated and solutions with higher fitness have higher probability to survive and to generate offsprings. A comprehensive overview of GA can be found in [Goldberg89]. The basic steps of GA are outlined in Algorithm 8.

Algorithm 8 GENETIC ALGORITHM (GA)

Generate an initial population
Iterative step:
    Select parents from the population
    Generate offsprings – crossover
    Apply mutation to the offsprings
    Make the offspring feasible with Algorithm 5 GREEDY
    Replace some of the population with the offsprings
Stop, if a termination criterion is fulfilled.

Let us now see the details of the algorithm. As in the case of Tabu Search, the possible solutions are n-dimensional 0-1 vectors representing the test sets the coverage of which are bigger than the given lower bound. The fitness is equivalent to the cost: the lower the cost of a solution is, the higher is its fitness.

The initial population is built at random. We generated random vectors and made them feasible with Algorithm 5.

The purpose of parent selection is to find two (or more) members of the population which can mate and generate offsprings. The two most widely used methods are the proportionate and the tournament selection. In the proportionate method, the individuals are selected based on a probability distribution which is proportional to their fitnesses. In the tournament selection, two groups of T members are picked out randomly from the population, and the individuals with the highest fitness in each group are assigned as parents. In our algorithm we employed the binary tournament selection (i.e.: $T = 2$). This method can be implemented very efficiently and experiments show (e.g.: for the Set Covering Problem in [Beasley96]) that in terms of solution quality, it is comparable to other methods.

Crossover originally meant that one or more crossover points are located and the segments of the parent vectors are swapped. Another possible technique to generate offsprings is the uniform crossover, when the i^{th} coordinate of the offspring ($i = 1, 2 \ldots, n$) equals that one of either the first or the second parent with equal probability. A more sophisticated version of this method is the fusion crossover, due to Beasley and Chu [Beasley96], where the probability is proportional to the fitness of the parents. We used this method in the algorithm.

Mutation means flipping (inverting) at random one or more coordinate values in the offspring. The number of the changed coordinates can be constant throughout the algorithm or it can vary. The classical, simplest method (see [Back93]) is to alter one

coordinate value in each iteration. The rationale behind the variable mutation rate is that mutation gets more and more important as the diversity of the population decreases. At the early iterations, when there are highly different solutions in the population, one should let crossover work, while towards the end of the algorithm, when GA has converged, higher mutation rates should be used to escape possible local minima. In [Beasley96], Beasley and Chu suggested a variable mutation rate as a function of the number of child solutions generated. Their function employs a few parameters the actual values of which are specified by the user or estimated through a test problem. In contrast to that static function which determines the mutation rate for each iteration at the beginning, we developed an *adaptive mutation rate* function which keeps account of the population's diversity and determines the number of mutated coordinates on the fly by, using the following formulas:

$$mutrate = \left\lfloor 1 + \frac{maxmut - 1}{t + 1} \right\rceil, \quad popdiv = \left\lfloor \frac{popmax - popmin}{avrgcost} \right\rceil \tag{5.13}$$

where $\lfloor a \rceil$ denotes the closest integer to a, $maxmut$ is a user defined parameter specifying the maximal mutation rate allowed, $popdiv$ describes the diversity of the population, $popmax$, $popmin$ denote the highest and lowest cost in the population and $avrgcost$ is the average cost of the test cases. Figure 5.1 shows the mutation rate as the function of $popdiv$ with $maxmut = 10$. This function determines the maximal number of changed coordinates. The actual mutation rate is a random number between 1 and $mutrate$.

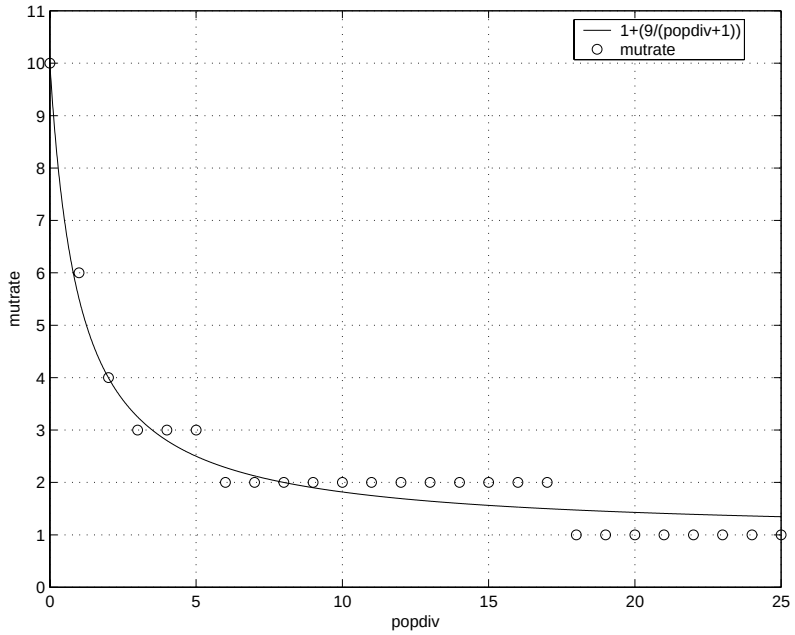

Figure 5.1: The adaptive variable mutation rate in GA

To keep the size of the population unchanged, one of its members has to be dropped out so that the new child solution can enter the population. We applied the steady-state replacement method to determine the solution that leaves the population: it is chosen at random among the members whose fitness is less than the average fitness of the population. With this method, the best solutions are always kept in the population. Special care must be taken with duplicate solutions, when the offspring is identical to a member of the population. To prevent the possibility of a population which contains only a few different solutions, duplicate offsprings are discarded and a new child solution is generated.

For termination, let i be the iteration counter and i^* denote the last iteration when the best solution in the population was improved. The algorithm is stopped if $i > min\{100\,000,\ i^* + 15\,000,\ 10i^*\}$ but at least 2 000 iterations have been done.

5.2.4 Simulated Annealing

Simulated Annealing (originally proposed by Kirkpatrick, Gelatt and Vecchi [Kirkpatrick83] and Cerny [Cerny85], a detailed description is to be found in [Aarts97]) is a randomized simple local search technique. An extensive empirical study of SA can be found in [Johnson89] and [Johnson91]. In Simulated Annealing one draws a solution $\mathbf{x}$ at random from the neighbourhood $V(\mathbf{x}_n)$ of the current solution $\mathbf{x}_n$ and accepts it as next solution $\mathbf{x}_{n+1}$ with probability $p(n)$. This probability depends on the difference between the solution value of $\mathbf{x}$ and $\mathbf{x}_n$ and the so-called *temperature,* $Temp(n)$, at iteration n. The most often used probability function is a Boltzmann-like distribution coming from thermodynamics. The scheme of Simulated Annealing is given in Algorithm 9.

Algorithm 9 SIMULATED ANNEALING (SA)

Initialization:

Generate initial solution x_1 with Algorithm 5 GREEDY

Let $c^* := cx_1$

Set initial temperature $Temp(1) := Temp_1$

Iterative steps $n = 1, 2, \ldots$:

x_n denotes the current solution

$x_{n+1} = x_n$

Let x be a random element of $V(x_n)$

Let $x_{n+1} = x$ with probability

$$p(n) = \min\left\{1, e^{\frac{cx_n - cx}{T(n)}}\right\}$$

if $cx_{n+1} < c^*$, then $c^* := cx_{n+1}$

Let $Temp(n+1) = q \cdot Temp(n)$

Stop, if the stopping criterion is met.

There are three things in this simple algorithm that must be specified: the cooling schedule, (i.e.: the way of determining $Temp(n)$), the neighbourhood structure, and the stopping criterion. The cooling schedule consists of setting the initial temperature and the method of decreasing it. We did the following: the temperature is decreased by factor q in each iteration from its starting value $Temp_1$. The possible parameter settings will be discussed in Section 6.4.

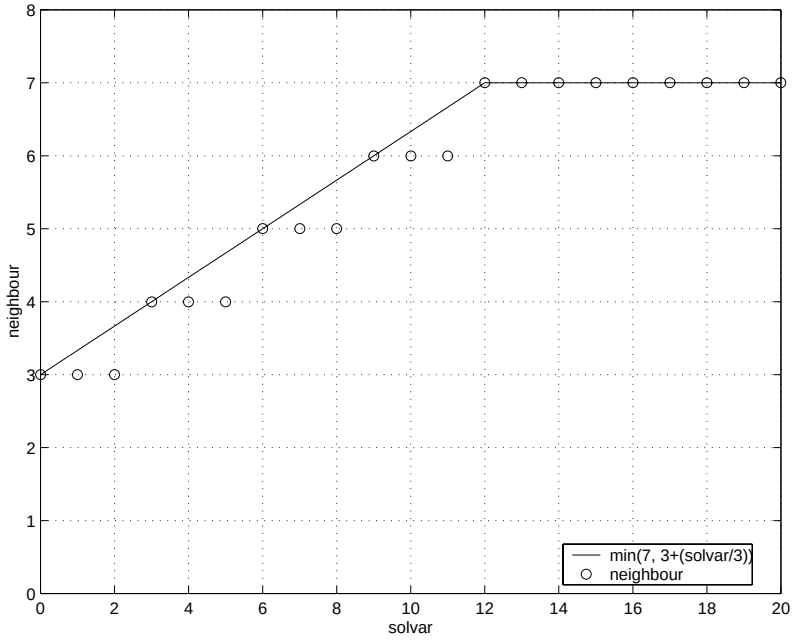

Figure 5.2: The adaptive variable neighbourhood structure in SA

The neighbourhood structure is more difficult. While the temperature is high, SA accepts more solutions and it is a good tactic to let it 'ramble' over the solution space so that it could find "hidden" places. On the other hand, when the temperature is low and the algorithm starts to freeze, less liberty should be given, as big changes are not acceptable anymore. This idea is reflexed by the neighbourhood structure. At the beginning of the algorithm, when there are big changes in the value of the iterative solutions, one should define a broader neighbourhood, and narrow it down as the variability decreases. Neighbourhood is defined through the number of coordinate values that can be flipped to obtain feasible solution $\mathbf{x}$. Broader neighbourhood means that $\mathbf{x}$ can differ from $\mathbf{x}_n$ in more coordinates. Thus, we developed a function, the *adaptive neighbourhood search* function which determines the maximum number, called *neighbour*, of coordinate values flipped. The actual number is set at random between 1 and *neighbour*. Once this number is known, the algorithm flips the value of that many randomly chosen coordinates in $\mathbf{x}_n$.

If the obtained vector is feasible, then it will be the random neighbour $\mathbf{x}$, otherwise the algorithm tries again with the same *neighbour* value. The value of *neighbour* depends on the relative variability *solvar* of the solutions in the recent iterations:

$$neighbour = min\left\{maxneigh, \left\lfloor \frac{v}{3} + minneigh \right\rfloor\right\}, \qquad solvar = \left\lfloor \frac{|currsol - prevsol|}{avrgcost} \right\rceil \tag{5.14}$$

where *maxneigh* and *minneigh* are the maximal and minimal values allowed, *currsol* is the value of the current solution ($\mathbf{cx}_n$ in iteration n), *prevsol* is the value of a previous solution (we used $\mathbf{cx}_{n-100}$) and *avrgcost* is the average cost of the test cases. Furthermore, $\lfloor a \rfloor$ means the greatest integer number inferior to a, and $\lfloor a \rceil$ denotes the closest integer to a. The values of *neighbour* for varying *solvar* in case of $maxneigh = 7$ and $minneigh = 3$ are shown in Figure 5.2.

The stopping criterion is threefold. The algorithm is stopped after a fix number (10^7) of iteration, if c^* has not been improved in the last 400 000 iterations or if there has been 200 000 unsuccessful trials to find a feasible random neighbour $\mathbf{x}$.

Chapter 6

Parameter settings in the solution methods

In this chapter, the parameters of the solution methods described in the previous chapter will be discussed, namely those of the B&B algorithm, TS, GA and SA. There is no parameter to set in the greedy algorithms, so we will not discussed them here.

The performance of the heuristic method can be strongly dependent on the parameters the algorithms use. We tested the heuristic methods with different parameter settings to find the one which ensures the best performance. On the other hand, this chapter justifies that our new adaptive techniques improve the quality of the heuristic solutions.

To compare the different parameter settings and evaluate the performance of the heuristic algorithms presented in Section 5.2, we applied them to different selection problems. We chose three important real-life protocols: an ISDN DSS1 layer 3 basic call protocol (ISDN), a GSM Phase 2 Mobile station layer 3 protocol (GSM) and the Intelligent Network Application Protocol (INAP). Their ATSs ([ETS403], [ETS607] and [ETS374]) are issued by the ETSI.

The ISDN test suite contains 668 test cases, the GSM has 324 while the INAP has 357. We derived the subpurposes and their relation to the test cases from the ATS using the automatic matrix filling program (Section 4.1.2) and obtained 192 subpurposes for ISDN, 218 for GSM and 243 for INAP.

We applied the meta-heuristic algorithms to the ISDN and the GSM protocols. We used two kinds of input vectors: a unit where all the test cases and subpurposes have cost and weight 1, and a random one, where the costs and weights are set at random between 1 and 10. As mentioned before, we deal only with the minimal cost problem. In this problem

the lower bound K for the coverage is given. For the ISDN test suite we chose K to be 20, 40, 60, ..., 180 and 100, 200, 300, ..., 1000 using unit and random coverage vectors, respectively. For the GSM test suite, in addition we chose $K = 200$ and $K = 1100$. These values can serve as examples of the overall performance of the methods. The details are summarized in Table 6.1.

Table 6.1: Problem details

Test suite	Subpurposes (k)	Tests (n)	Input	$cov(TS)$	$c(TS)$
ISDN	192	668	UNIT	192	668
			RANDOM	1075	3765
GSM	218	324	UNIT	218	324
			RANDOM	1184	1754

6.1 Branch and Bound

We employed two commercial ILP solvers: CPLEX MIP Solver ([CPLEX]) and IBM's OSLMSLV Solver ([OSLMIP]). They use powerful B&B algorithm completed with preprocessing and heuristic steps. In both cases we limited the number of nodes of the B&B tree to 30 000 to avoid very long running times of the algorithm. Furthermore, in CPLEX we set the backtracking parameter to 0.25 and in OSLMSLV we stopped the algorithm after two hours. We also raised the memory space available for solution.

In Chapter 8, the results of CPLEX will be used to optimize the selected test set in real-life applications. In this chapter and Chapter 7 the B&B solutions obtained by using OSLMSLV serve as a basis for comparison of different heuristic algorithms and different parameter settings.

Some comments on OSLMSLV should be made here. Its log-file reports every integer solution found as optimal, even if the Branch and Bound tree is pruned before being searched through entirely. What is more inexplicable, is that in some cases we got 'optimal' solutions after a complete search which were worse than the otherwise known – really optimal – solutions. That is why, in each case we treat the solution of OSLMSLV as only a best integer solution and not as an optimal one.

6.2 Tabu Search

The RTS algorithm was presented in Section 5.2.2. The only parameter to be set is the neighbourhood sampling coefficient nb. Table 6.2 and 6.3 present the results of our experiments with different values of nb for the GSM protocol in All or Nothing (ALL) and Step and Jump (STEP) models for unit and random input vectors, in case of $K = 100$ and $K = 600$, respectively. We also made experiments with the ISDN protocol and other coverage bounds, and found their results similar. The numbers represent the average relative deviation from the B&B solution and the average execution time for 25 trials in all instances. The results are divided into two tables because the behaviour of the RTS algorithm with respect to varying the neighbourhood coefficient differs significantly in case of unit and random input vectors.

Table 6.2: Comparison of adaptive neighbourhood sampling for unit input

Coverage	average relative deviation %						
model	$nb = 1$	$nb = 2$	$nb = 4$	$nb = 7$	$nb = 11$	$nb = 16$	$nb = 22$
STEP	0.54	0.45	0.85	0.99	1.30	0.94	0.85
ALL	0.66	1.23	0.84	1.27	1.76	1.14	2.11
Average	0.60	0.84	0.85	1.13	1.53	1.04	1.48
Coverage	average execution time (sec.)						
model	$nb = 1$	$nb = 2$	$nb = 4$	$nb = 7$	$nb = 11$	$nb = 16$	$nb = 22$
STEP	253.44	258.76	188.20	109.36	74.32	54.56	43.36
ALL	200.08	193.48	130.76	87.04	57.76	44.20	33.60
Average	226.76	226.12	159.48	98.20	66.04	49.38	38.48

Table 6.3: Comparison of adaptive neighbourhood sampling for random input

Coverage	average relative deviation %						
model	$nb = 1$	$nb = 2$	$nb = 4$	$nb = 7$	$nb = 11$	$nb = 16$	$nb = 22$
STEP	22.39	28.92	22.32	38.69	29.03	25.59	23.80
ALL	8.64	8.79	8.54	8.32	10.11	9.14	8.96
Average	15.52	18.86	15.43	23.51	19.71	13.37	16.38
Coverage	average execution time (sec.)						
model	$nb = 1$	$nb = 2$	$nb = 4$	$nb = 7$	$nb = 11$	$nb = 16$	$nb = 22$
STEP	540.60	504.00	541.76	499.72	522.04	535.64	547.76
ALL	305.76	317.32	314.68	305.96	274.32	290.04	300.72
Average	423.18	410.66	428.22	402.84	398.18	412.84	424.24

For unit vectors (see Table 6.2) the quality of the solution deteriorates noticeably, though not steadily, as fewer neighbours are sampled. The execution time, on the other hand, decreases significantly. This means that choosing low but not minimal value for nb

will ensure relatively good quality solutions and the execution time can be reduced. We set $nb = 5$ in order to match the execution time of the other meta-heuristics.

In case of random input vectors (Table 6.3), neither the solution nor the time are affected by changing the value of parameter nb. Hence, setting $nb = 5$ is not worse than any other value.

6.3 Genetic Algorithm

Section 5.2.3 presents the GA algorithm and the adaptive mutation rate technique. The parameters to set are the size of the population and the maximal mutation rate allowed in the mutation rate formula (5.13).

The population size had no effect on the solution quality, provided that it was not too small. On the other hand, the bigger the population was, the longer the algorithm took, so we kept the population small, consisting of 50 members.

Table 6.4: Comparison of various mutation rates

Problem (model/input)	Average relative deviation %		
	$mutrate = 1$	$mutrate = 10$	Variable rate
STEP/RANDOM	0.65	0.76	0.67
STEP/UNIT	1.68	1.63	1.73
ALL/RANDOM	-1.27	-1.27	-1.39
ALL/UNIT	0.53	0.75	0.44
Average	0.39	0.46	0.36

We set the maximal mutation rate to 10, and made experiments to judge its performance. In our computational study, 10 trials of GA were made in each combination of All or Nothing (ALL) and Step and Jump (STEP) models, unit and random input vectors, and for all listed lower bounds K. Table 6.4 reports the average relative deviation from the Branch and Bound results for the GSM protocol, for constant mutation rates $mutrate = 1$ and 10 and the variable mutation rate determined with (5.13), setting $maxmut$ to 10. The results show that on average the performance of the variable mutation rate is the best, although in some cases the other two outperformed it. The execution times were very similar in all cases. In our further experiments we used the variable mutation rate.

6.4 Simulated Annealing

As mentioned in Section 5.2.4, where the Simulated Annealing algorithm is presented, we will discuss the cooling schedule and the neighbourhood structure in this section.

6.4.1 Cooling schedule

The cooling schedule consist of setting the initial temperature ($Temp_1$) and the factor q with which the temperature is decreased in each step. To find the best initial temperature and cooling factor we set

$$Temp_1 = 2^{l-5} \text{ for } l = 1, 2, \ldots, 15 \text{ and } q = 1 - 10^{-\frac{m}{3}} \text{ for } m = 1, 2, \ldots, 15$$

and for all the 225 combinations we let SA run 10 times for ISDN protocol in minimal cost problem, All or Nothing model for $K = 95$ coverage bound. The resulting test set in run i is denoted T_i, while the execution time of this run is $time_i$. Figure 6.1 shows the graph of the *average solution*:

$$avrgsol = \frac{1}{10} \sum_{i=1}^{10} c(T_i), \quad i = 1, \ldots, 10.$$

in the 225 possible cases as well as the projection of this graph. In this latter one, the darker area represents better solutions.

The results show that if the initial temperature is bigger than 2 ($l \geq 6$) and q is at least 0.999 ($m \geq 9$), then we get good quality solutions with high probability.

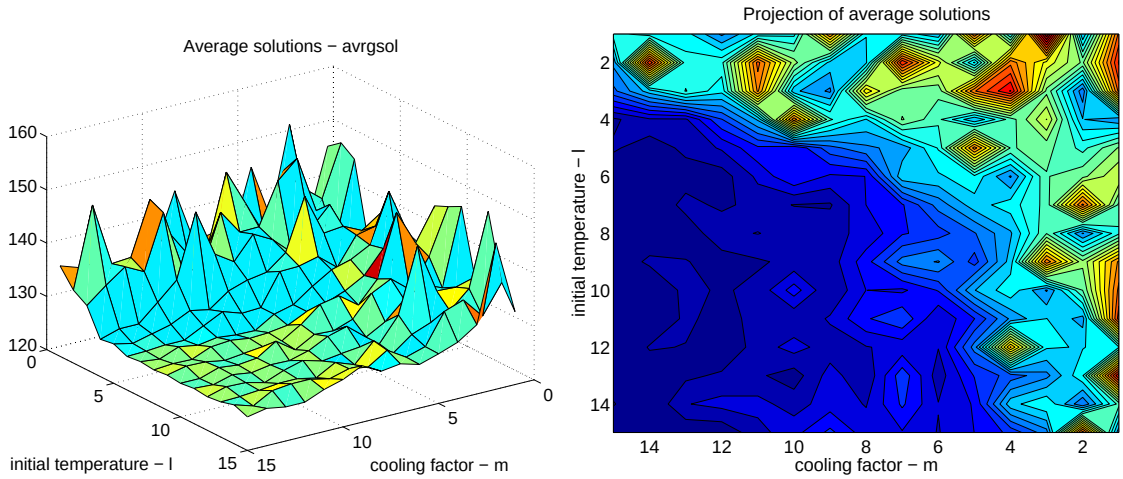

Figure 6.1: The average solutions and their projections

Figure 6.2 presents the graph of the *average execution time*:

$$avrgtime = \frac{1}{10}\sum_{i=1}^{10} time_i, \;\; i = 1, \ldots, 10.$$

in the 225 possible cases as well as the projection of this graph. In this latter one, the darker area represent shorter execution time.

The execution time increases linearly as l grows and exponentially as q approaches 1. If m is less than 10-12, that is $q \leq 0.999 \ldots 0.9999$, then the execution time is acceptable, while for bigger cooling factors, it becomes too much.

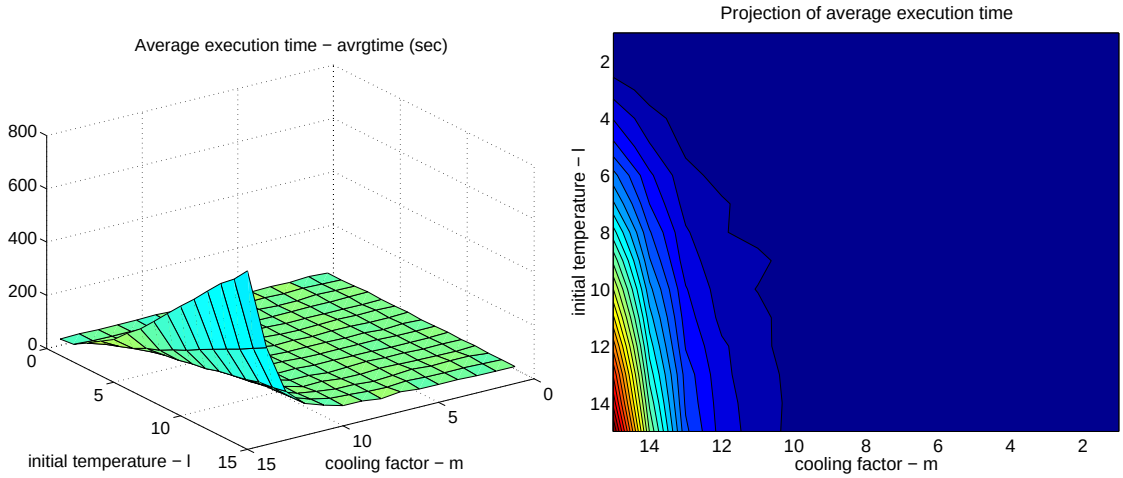

Figure 6.2: The graph of the average execution times and its projection

Let us introduce a function measuring the *efficiency* of the cooling schedule by taking into account both the achieved solution and the execution time. For this purpose, let the efficiency to be defined as the weighted sum of the standardized average solution and the standardized average execution time:

$$efficiency = \alpha_1 \frac{avrgsol - minsol}{maxsol - minsol} + \alpha_2 \frac{avrgtime - mintime}{maxtime - mintime},$$

where $minsol = \min(c(T_i) \mid i = 1, \ldots, 10)$, $maxsol = \max(c(T_i) \mid i = 1, \ldots, 10)$ are the minimum and maximum solution costs of the 10 trials. In the same way we got the $mintime = \min(time_i \mid i = 1, \ldots, 10)$ and $maxtime = \max(time_i \mid i = 1, \ldots, 10)$ values for the execution times of the algorithm. The efficiency function and its projection for the 225 parameter settings with $\alpha_1 = \alpha_2 = 1$ is presented in Figure 6.3.

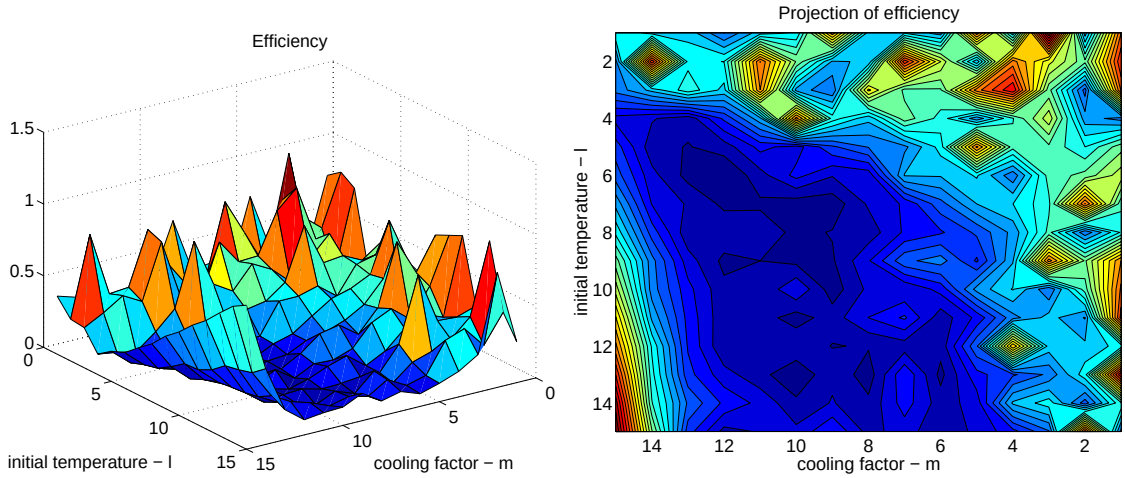

Figure 6.3: The efficiency function and its projection

The darker a region is in the projection, the more efficient is the cooling schedule. We chose the initial temperature and the cooling factor from the darkest region by setting $Temp_1 = 128$ $(l = 12)$ and $q = 0.9999$ $(m = 12)$.

Note that the reliability of the results achieved by a stochastic heuristic method can be characterized by their standard deviation. That is to say, very often it is not enough to get a good average solution after a few trial, the variance of these solution should also be low. Knowing the standard deviation allows us to set the level of confidence we have for the method. Figure 6.4 shows the standard deviation of the 10 solutions. It is very similar to the average solution presented in Figure 6.1, meaning that choosing a cooling schedule with good average performance has low variance too.

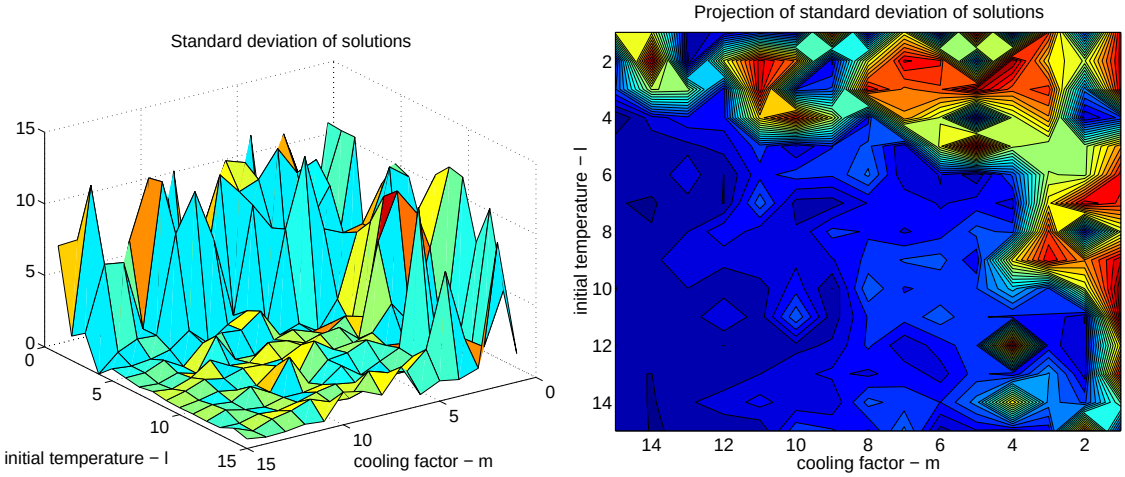

Figure 6.4: The standard deviations of the solutions and their projections

6.4.2 Neighbourhood structure

We compared the fixed and variable neighbourhood structure with 10 trials for the same problems as in case of GA. We tried out the constant $neighbour = 1$ and $neighbour = 3$ and the variable structure with setting $maxneigh = 7$ and $minneigh = 3$ in (5.14). Table 6.5 shows the average relative deviation from the Branch and Bound solutions for ISDN protocol. On average, our adaptive neighbourhood structure outperformed the fixed structures, but in some cases (especially for unit input vectors) we got better solutions with $neighbour = 3$. The classical method, where exactly one coordinate value is flipped in each iteration ($neighbour = 1$), was much worse than the other two methods with bigger changes. On the other hand, it took less time (around 70 seconds) than the others (the average execution time was around 200 seconds), but that did not compensate for its poor performance. We chose the variable neighbourhood structure to be used in our further experiments.

Table 6.5: Comparison of various neighbourhood structures

Problem	Average relative deviation %		
(model/input)	$neighbour = 1$	$neighbour = 3$	Variable structure
STEP/RANDOM	12.34	5.12	4.34
STEP/UNIT	16.04	3.07	3.31
ALL/RANDOM	8.01	5.00	4.43
ALL/UNIT	3.12	0.89	1.02
Average	9.87	3.52	3.27

Chapter 7

Experimental results on heuristics

We presented heuristic algorithms in Section 5.2 and in this chapter the experimental results on them will be presented. The test protocols (ISDN, GSM and INAP) and the input vectors on which the performance of the algorithms was evaluated are presented in Chapter 6.

7.1 Greedy algorithms

To analyze the performance of the greedy algorithms we examined Abstract Test Suites of the GSM and INAP.

In greedy algorithm in contrast to meta-heuristics, we used only unit cost and coverage vectors as we had no reason for differentiating between the tests and subpurposes. The results are for the minimal cost problem in One is Enough, Step and Jump and All or Nothing models for coverage bound $K = 1, 2, \ldots, cov(SP)$. The details are summarized in Table 7.1.

Table 7.1: Problem details

Test suite	Subpurposes (k)	Tests (n)	Input	$cov(TS)$	$c(TS)$
GSM	218	324	UNIT	218	324
INAP	243	357	UNIT	243	357

The results of different algorithms were compared with each other and with the best integer solution obtained by applying OSLMSLV. This commercial IP solver found the optimal solution for each value of K in the One is Enough model, while in the other two models it was often able to provide only feasible solutions within the time limit.

7.1.1 Results of the greedy algorithms

We used the One is Enough (ONE), Step and Jump (STEP) and All or Nothing (ALL) model in minimal cost optimization problem. So for the GSM and INAP test suites and the three coverage models we solved six different optimizations. In each cases, we present the results of ADD, DROP, ADD-DELTA and DROP-DELTA greedy algorithms (see Section 5.2.1) as compared to the best integer solutions obtained from OSLMSLV. The algorithms were implemented in MATLAB.

In case of the One is Enough, model algorithms ADD-DELTA and DROP-DELTA give solutions very close to the best integer one both for GSM (see Appendix A Figure A.1) and for INAP (Figure A.4). We can also observe that the simple algorithm ADD produces very poor results in the One is Enough model. The results for the Step and Jump model are presented in Figure A.2 for GSM and in Figure A.5 for INAP in Appendix A. In these cases, algorithm DROP gave the worst results compared to the other three methods. In the All or Noting model the results of the greedy algorithms are very similar to those in the Step and Jump model. Figure A.3 and Figure A.6 present the results for GSM and for INAP.

Table 7.2 gives a summary of the results presented in Figures A.1-A.6. For each greedy algorithm, the average relative gap between the best integer solution obtained from the OSLMSLV and the greedy solution is shown. The average is calculated over the varying coverage bounds.

protocol	model	ADD	DROP	ADD-DELTA	DROP-DELTA
GSM	ONE	296%	92%	4.25%	4.72%
	STEP	66%	118%	14%	42%
	ALL	67%	105%	16%	82%
INAP	ONE	157%	86%	2.9%	5%
	STEP	78%	122%	9%	24%
	ALL	78%	110%	10%	50%

Table 7.2: The average relative gap between the solution of the heuristics and the commercial solvers

7.1.2 Examining the results

As it is seen from the results, the ADD-DELTA algorithm was the best with respect to the average relative gap, though other algorithms sometimes gave better solutions for specific bounds. For some specific coverage bounds ADD-DELTA could even find better solutions

than the commercial software. In the One is Enough model the biggest (absolute) gap between the best OSLMSLV and the ADD-DELTA solutions was 3, so in this case this greedy algorithm practically found the best solutions achievable by using a commercial software.

It is important to observe that we got very similar results for the four different greedy algorithms when we applied them to the GSM and INAP protocols. This fact suggests that we can make general statements about the performance of the algorithms independently of the protocol.

7.2 Reactive Tabu Search, Genetic Algorithms and Simulated Annealing

We applied the meta-heuristic algorithms to the ISDN and the GSM protocols in the minimal cost problems, All or Nothing and Step and Jump models, because these coverage models seemed the most relevant for the ISDN and GSM test suites. Of course, the other models can be very suitable for other protocols. The cost and weight vectors we used we presented in Chapter 6.

7.2.1 Results of the Reactive Tabu Search, Genetic Algorithms and Simulated Annealing

In this section we present the comparison of the Reactive Tabu Search, the Genetic Algorithms and the Simulated Annealing algorithms. The algorithms were coded in C++ and run on a PC with a PentiumII MMX 375 MHz processor. We solved the minimal cost problem for the ISDN and GSM protocols, using unit and random input vectors (see Table 6.1 for the details) in All or Nothing and Step and Jump models.

We ran 10 trials of all algorithms. The results for the ISDN and GSM protocols are shown in Table A.1 and Table A.2 in Appendix A. We give the lower bound for the cost (K), the Branch and Bound solution (B&B) obtained from OSLMSLV, the minimum (min), maximum (max) and average (avg) value of the solutions in the 10 trials for the Genetic Algorithm and for Simulated Annealing. Furthermore, the average relative deviations (avg σ) from the B&B solutions and the average execution times (user time in seconds) of the algorithms are also given.

The detailed results of RTS are not presented because they were worse than those of SA or GA in all aspects. This fact is also apparent in Table 7.3 and Table 7.4, which give the overall average relative deviations from the B&B solutions and the overall average execution times in the different problems for all the three meta-heuristics.

Table 7.3: Average results of RTS, GA and SA for ISDN

Problem (model/input)	RTS		GA		SA	
	avg σ %	avg time	avg σ %	avg time	avg σ %	avg time
STEP/RANDOM	187.70	727.05	2.73	370.33	4.34	85.88
STEP/UNIT	5.78	325.99	3.38	562.71	3.31	70.78
ALL/RANDOM	20.68	570.11	1.99	263.55	4.43	62.18
ALL/UNIT	6.11	266.72	1.92	257.48	1.02	79.93

Table 7.4: Average results of RTS, GA and SA for GSM

Problem (model/input)	RTS		GA		SA	
	avg σ %	avg time	avg σ %	avg time	avg σ %	avg time
STEP/RANDOM	15.71	451.25	0.67	148.89	1.56	66.87
STEP/UNIT	3.83	169.24	1.73	202.49	1.39	62.71
ALL/RANDOM	4.86	248.44	-1.39	117.19	-0.16	54.26
ALL/UNIT	4.02	132.80	0.44	139.70	1.23	55.45

7.2.2 Examining the results

The RTS algorithm results in significantly worse solutions than the other two meta-heuristics. This is most evident when the random input vectors are used, especially for the ISDN protocol. While for unit input the RTS finds comparable solutions in acceptable time, its performance for random input is far from satisfactory.

Examining the results, we can observe that GA performs definitely better than SA for random input vectors, while the average performance of SA is superior for unit input vectors, especially for the ISDN protocol. For particular K values, however, GA can often provide better solutions.

The execution time of SA was significantly smaller than that of GA. This implies that more SA trials can be executed in an acceptable time period, thus possibly better solution can be obtained. One iteration step in SA is much simpler and faster than in GA, but GA converge faster to the optimal solution than SA.

It is worth considering which is a better tactic: to let a heuristic algorithm run for a long time or to limit the execution time while doing more trials, and finally choosing the best solution. As the SA algorithm is so fast, several trials can be executed with for the same bound.

On average, the results of SA and GA are at most 5 % worse than those obtained from OSLMSLV. In cases where the average relative deviation is worse than 5 %, the minimal heuristic solutions are always below this gap. For several instances the meta-heuristics could provide us with the same solutions as OSLMSLV. GA found the same minimum as OSLMSLV in 21 cases out of 38 for the ISDN protocol and in 28 cases out of 42 for GSM. As for SA, the numbers are 14 out of 38 and 22 out of 42. In some cases the heuristic methods gave better solutions than the commercial software (these are typed in bold in Table A.1 and Table A.2). This is the most noticeable for GSM in the All or Nothing model, using random input vectors, where even the average results of GA and SA are lower than that of OSLMSLV.

These observations can be summarized as follows. In terms of solution quality GA seems to have the most desirable performance, while SA possesses the best time characteristics. The RTS algorithm cannot provide the same quality as the other two meta-heuristics. The average relative deviations of GA and SA from the B&B solutions are less then 5 %. The best results from 10 trials can, of course, provide even better approximations. In the majority of the instances GA and SA find at least as good solution as OSLMSLV.

The results imply that the heuristic algorithms produce acceptable solutions, which are sufficiently close to the optimum in real-life applications. So it is not necessary to use expensive and complex commercial IP solvers because these heuristic algorithms can give us good approximative solutions without any drawback of the commercial softwares (e.g.: CPLEX, OSLMSLV). Furthermore, the implementation of the algorithms is relatively simple, the program code is portable, it does not require licenses, and the algorithms are adjustable to the concrete tasks.

Chapter 8

Real-life applications

We already presented three different applications of the test selection method on an ISDN ([ETS403]), a GSM ([ETS607]) and an INAP ([ETS374]) test suite in Chapter 6 and 7, where we applied different heuristic algorithms. There we focused on the heuristic test selection algorithms because the subpurpose-test case incidence matrixes were so large for these protocols that the available ILP algorithms (B&B) often failed to find the exact optimum. This is the reason why we presented those applications in the chapter dealing with heuristics. Now we present two other applications of our test selection method where the size of the problem lets us use methods that search for exact optimal solutions.

In the first one, we study a test suite which has a special property: the standard of ISDN primary rate access DSS1 Layer 2 protocol ([TBR4]) explicitly lists the conformance requirements and the related test cases as mentioned in Section 4.1.1. In such a case, the creation of the requirement-test case incidence matrix is very simple; we can simply fill it by hand.

In the second application we specialize our test selection process for the function test methodology. As in conformance testing, our goal was to develop a theoretical method that helps testers to select test cases and makes function testing more economical and reliable. We developed the test selection method originally for conformance testing, but we found that it is important to examine how it could be applied to other methodologies, such as function testing.

8.1 ETSI TBR4 application

As we mentioned in Section 4.1.1, the requirement-test case incidence matrix of the ISDN primary rate access DSS1 Layer 2 protocol ([TBR4]) can be filled manually. By simply reading the standard, we found 27 conformance requirements ($k = 27$) and 52 test cases ($n = 52$) together with their relation. We used a unit weight vector, as there was no reason to differentiate conformance requirements.

We defined three different cost vectors:

- The unit cost vector $\mathbf{c}_1$. This can be used if we are interested in only the number of the selected test cases; for example if their costs are all equal.
- The second cost vector ($\mathbf{c}_2$) is based on the timers contained by the test cases. We estimated $\mathbf{c}_2$ using the sum of the default times of the timers in the test cases. This time can be an upper bound for the execution time of T_j. The cost values used are as follows:

$$\begin{aligned}\mathbf{c}_2 = \quad & (6,3,33,3,3,8,4,5,3,9,6,35,3,13,2,4,34,6,6,2,34,5,3,5,2,5,\\ & 1,8,33,7,8,3,2,1,2,35,4,2,2,3,2,4,6,2,2,2,33,33,31,6,1,2)\end{aligned}$$

- The definition of the third cost vector ($\mathbf{c}_3$) is based on the assumption that the main time consuming steps of testing are the preparation and, in case of fault, the search for its cause. That is why we added a constant value (100) to every cost in c_2 referring to this time, so $\mathbf{c}_3 = \mathbf{c}_2 + 100$.

To solve the ILP problem presented in Section 5.1, we used the CPLEX software ([CPLEX]) with parameter setting described in Section 6.1.

8.1.1 Minimal cost problems

1. **All or Nothing model:** We examined all the three cost vectors. The results of the optimization problem (5.7), for $K = 1, \ldots, 27$ and for the three cost functions are shown in Figure B.1, Figure B.2, Figure B.3 in Appendix B.

   We can see in all cases that when increasing the coverage bound, the cost of the optimal test set does not increase linearly. This means that when using our method, we can obtain better (shorter in time) selected test sets, than we would get if we chose them at random. In fact, our method gives us the best possible test set within

the constraints.

The figures also show that the c_2 cost vector has the best characteristics, that is, the graph increases with the slowest rate (Figure B.2). This is because the variation of the cost values is the largest in this case. It means that we can reduce the cost with only a small loss in coverage. The cost jumps when, in order to reach the required coverage, it is necessary to execute the test cases with bigger cost.

Where the variation in the cost vector is less, as in case $\mathbf{c}_3$ and especially in $\mathbf{c}_1$, the graph is smoother.

2. **Linear model:** The cost of the optimal test set for $K = 1, \ldots, 27$ is shown in Figure B.4, Figure B.5 and Figure B.6.

3. **One is Enough model:** Figure B.7, Figure B.8 and Figure B.9 show the optimal cost in the One is Enough model using cost vectors $\mathbf{c}_1$, $\mathbf{c}_2$ and $\mathbf{c}_3$.

8.1.2 Maximal coverage problems

1. **All or Nothing model:** When we are looking for a maximum coverage test set, we have an upper bound for the cost (L). Different cost vectors have different maximal upper bounds (L_{max} =52, 477 or 5677). We present the three graphs for this model in Figure B.10, Figure B.11 and Figure B.12.

2. **Linear model:** Figure B.13, Figure B.14 and Figure B.15 show the result of the maximal coverage problem for the Linear model using the three cost vectors.

3. **One is Enough model:** The results of the maximal coverage problem for the One is Enough model using the different cost vectors are shown in Figure B.16, Figure B.17 and Figure B.18. For example, we can observe that the graph in Figure B.17 reaches the maximal coverage at $L = 160$, so in this model only the third of the total cost is enough for the whole coverage.

8.2 Function test application

We applied the selection method to the BGC Attendant ISDN-E Connection's ([BGC]) tests which are real-life test objects and follow Ericsson's internal function test methodology. This test suite was executed previously by testers at Ericsson, so the knowhow and test result were provided. We compared the solution with an experienced tester's selection.

8.2.1 Input data and coverage models

The input data was created by hand (see Section 4.1.1). The test object has 88 test cases. We defined 85 subpurposes (i.e.: the subpurpose-test case incidence matrix has a size of 85×88). We used the unit cost vector because in practice we are usually interested in only the number of the test cases, as the time spent on preparing and executing different test cases is very similar. Of course, if we know in advance that the effort a test case needs is significantly different, then this can be expressed in choosing different cost.

We defined three different weight vectors. The first is a unit, all-one vector ($\mathbf{w}_1$). This can be used if we do not want to distinguish between the subpurposes. In the other two cases, we divided the subpurposes into 10 groups depending on their importance and gave weight value 10 to the test cases in the most important group, 9 to those in the second important group, and so on. The second ($\mathbf{w}_2$) and the third ($\mathbf{w}_3$) weight vector differ in the importance of some subpurposes:

$$\begin{aligned}\mathbf{w}_2 = \quad & (5,6,7,8,4,4,3,3,3,4,3,4,4,5,6,7,8,4,4,3,3,3,4,3,4,4,5,4,7,7,4,3,5,5,5,\\ & 4,7,7,4,3,5,5,8,9,9,8,8,8,8,8,8,8,8,8,8,10,10,10,9,9,8,6,5,6,6,7,7,7,7,\\ & 10,6,68,7,7,6,6,7,7,7,9,9,9,9,9,9)\end{aligned}$$

$$\begin{aligned}\mathbf{w}_3 = \quad & (5,6,7,8,4,4,3,3,3,4,3,4,4,5,6,7,8,4,4,3,3,3,4,3,4,4,5,4,7,7,4,3,5,5,5,\\ & 4,7,7,4,3,5,5,8,9,9,8,8,8,8,8,8,8,8,8,8,10,10,10,9,9,8,6,5,6,6,7,7,7,7,\\ & 10,6,6,8,7,7,6,6,7,7,7,1,1,1,1,1,1).\end{aligned}$$

We used the One is Enough model in optimization. In this model the purpose is considered to be entirely checked if at least one selected test case checks it. We do not obtain bigger coverage if more selected TCs check it. In the Linear model, however, we do. That is why we have carried out the optimization in Linear model as well. We used both the maximal coverage and minimal cost problems. So for the 3 weight vectors, 2 coverage models and the 2 optimization problems we solved 12 different optimizations.

8.2.2 Results of the test selection

To solve the ILP problems presented in Section 5.1, we used the CPLEX software ([CPLEX]) with parameter setting described in Section 6.1.

We gave the test object to an experienced tester and asked him to select those test cases he would execute. He selected 34 test cases out of 88. We calculated the coverage of these test cases using the different weight vectors and coverage models.

Figures C.1-C.6 in Appendix C show the costs of the the optimal test sets for varying coverage bounds (K) in the **minimal cost problem** for Linear and One is Enough models using weight vectors $\mathbf{w}_1$, $\mathbf{w}_2$ and $\mathbf{w}_3$.

To compare the cost of the optimal test set achieved by our method and that of the test set selected in the test laboratory, we used Linear and One is Enough models. The solutions we got were better in every case than the expert's solutions. We indicate the expert's and the optimal solutions with arrows in every figure (Figure C.1-C.6). Table 8.1 summarizes the results and gives the percentage of the cost increase. The cost is equal to the number of selected test cases, as we used unit cost vectors.

	Minimal cost problem					
	Linear			One is Enough		
	$\mathbf{w}_1$	$\mathbf{w}_2$	$\mathbf{w}_3$	$\mathbf{w}_1$	$\mathbf{w}_2$	$\mathbf{w}_3$
calculated cov. bound (K)	32.8	223.8	198.8	45	309	285
expert's solution	34	34	34	34	34	34
optimal solution	22	20	18	22	17	14
gain in cost	35%	41%	47%	35%	50%	58%

Table 8.1: Results of the minimal cost problems

Figures C.7-C.12 in Appendix C show the coverages of the the optimal test sets for varying cost bounds (L) in the **maximal coverage problem** for Linear and One is Enough models using weight vectors $\mathbf{w}_1$, $\mathbf{w}_2$ and $\mathbf{w}_3$.

	Maximal coverage problem					
	Linear			One is Enough		
	$\mathbf{w}_1$	$\mathbf{w}_2$	$\mathbf{w}_3$	$\mathbf{w}_1$	$\mathbf{w}_2$	$\mathbf{w}_3$
cost bound (L)	34	34	34	34	34	34
expert's solution	32.8	223.8	198.8	45	309	285
optimal solution	47	338.1	323	57	426	402
gain in coverage	43%	52%	62%	27%	38%	41%

Table 8.2: The results of the maximal coverage problems

Here we also indicate the expert's and optimal solutions with arrows in every presented figure (Figure C.7-C.12). Table 8.2 gives a summary of the results and shows the percentage of the gain in coverage.

Note that in every presented case (see Figures C.1-C.12) the solutions of the expert are near to the linear line connecting the less and biggest optimal costs or coverages. This suggests that the selection of experts can be modeled with this linear approximation.

Chapter 9

Conclusions

In this chapter we summarize and conclude our work and raise some idea for the future work.

9.1 Summary of the thesis

We have presented the theory and practice of a new selection method which selects test cases from an Abstract Test Suite. We have performed experiments to explore its efficiency. The results show that the method can reduce testing time. We have applied the method to fairly complex, widely used real-life protocols, the ISDN DSS1 layer 3 basic call protocol ([ETS403]), the GSM Phase 2 Mobile station layer 3 protocol ([ETS607]), the Intelligent Network Application Protocol ([ETS374]), the ISDN primary rate access DSS1 Layer 2 protocol ([TBR4]) and Ericsson's internal function test object the BGC Attendant ISDN-E Connection's ([BGC]). We find our method to be empirically promising but further experiments should be conducted for other protocols.

After a thorough introduction (Chapter 1) we outlined the theory of automation in the test selection problem in Chapter 2. Using this theory we showed a method for formulating the test selection problem in real-life conformance testing as a Mathematical Programming problem in Chapter 3. Before the problem solution methods, we put the theory into practice by presenting the test selection process in Chapter 4. In Chapter 5, we presented the Integer Linear Programming formulation of the test selection problem and introduced heuristic algorithms. These included five greedy algorithms and three more complex meta-heuristics: Reactive Tabu Search, Genetic Algorithms and Simulated Annealing.

The two simpler greedy methods, ADD and DROP (Algorithm 1 and Algorithm 2) can describe the heuristics used by test laboratories. By modifying them we got Algorithm ADD-DELTA and Algorithm DROP-DELTA that were able to provide comparable results to those of commercial softwares (see Chapter 7). The fact that these simple, fast and easily implementable algorithms (especially ADD-DELTA) could give us very good results which are definitely better than those obtained in practice, means that by applying them to test selection problem conformance testing can be made much more economic. On the basis of these four greedy algorithms, we developed Algorithm GREEDY, to be used in the meta-heuristic algorithms.

The metaheuristics we developed for the conformance test selection problem were based on known methods but we enhanced them with new techniques. That is to say, we used adaptive neighbourhood sampling in Reactive Tabu Search, we designed a new adaptive variable mutation rate function for Genetic Algorithms and an adaptive neighbourhood structure for Simulated Annealing to improve their performance.

When applying meta-heuristics, setting their parameters is very important and can significantly affect their performance. A thorough discussion of parameter settings can be found in Chapter 6.

Our experiments (see Chapter 7) showed that GA and SA can give comparable results to those obtained from a commercial software. The Tabu Search approach, however, was not able to achieve this quality, despite the extensive trials with different algorithms and parameter settings.

Finally, in Chapter 8, two real-life applications were presented. We demonstrated that our test selection method is easily applicable even in cases if the test suite is different than the common standardized ones (Section 8.1), or the test method differs from standardized conformance testing (Section 8.2).

9.2 Further research

We strongly believe that our method can be successfully applied only if it involves a fully automated test selection process. At the present, we choose PDUs to be subpurposes. This means that the PERL program, which is able to fill up the subpurpose-test case incidence matrix automatically, is based on the PDUs in TTCN version 2 machine processable form. In other words, we chose the second abstract data level (see Section 2.3), which is the PDU constraint level in TTCN version 2 for implementation.

Thus, a possible research area for the future could be the implementation of all abstract data levels like the ASP and/or PDU type level or the parameters of ASPs and/or PDUs. The implementation of these levels in PERL would require a whole TTCN syntax checking, leading to the implementation of a complete TTCN parser.

Of course, it is very promising to update the PERL program from TTCN version 2 to TTCN version 3. It seems that this new version of TTCN will change the old TTCN version 2, but at this moment there are not enough TTCN-3 test suites available. Therefore, we have not started to upgrade PERL program, because we could not find whole test suites in sufficient numbers. If there are several test suites available for real life applications, this subject could come to the forefront of research.

Appendix A

Additional results on heuristics

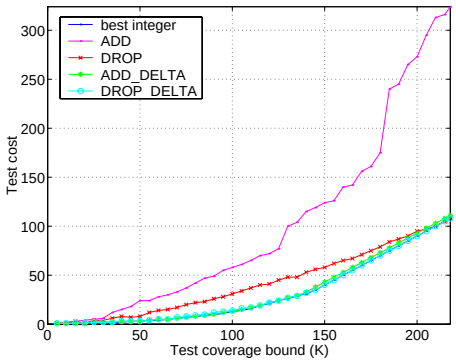

Figure A.1: Greedy algorithms for GSM in minimal cost problem, One is Enough model

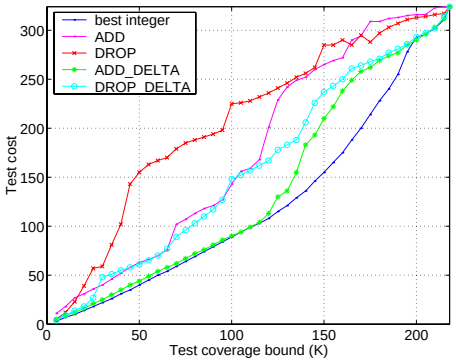

Figure A.2: Greedy algorithms for GSM in minimal cost problem, Step and Jump model

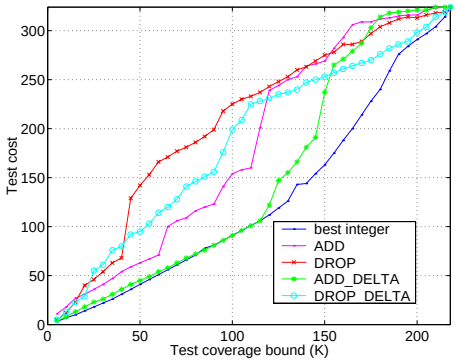

Figure A.3: Greedy algorithms for GSM in minimal cost problem, All or Nothing model

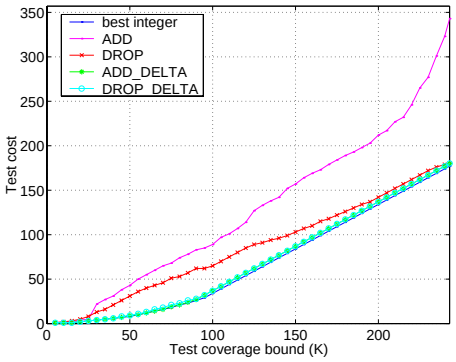

Figure A.4: Greedy algorithms for INAP in minimal cost problem, One is Enough model

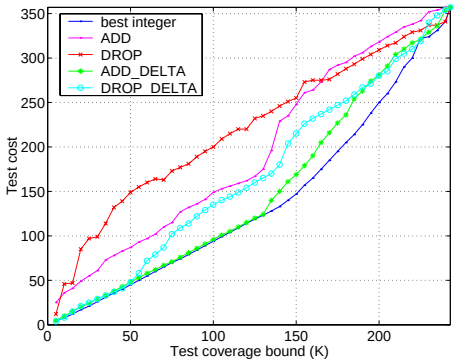

Figure A.5: Greedy algorithms for INAP in minimal cost problem, Step and Jump model

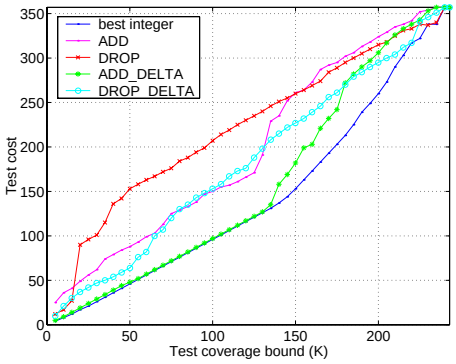

Figure A.6: Greedy algorithms for INAP in minimal cost problem, All or Nothing model

Table A.1: Computational results of GA and SA for ISDN

Problem (model/input)	K	B&B	GA					SA				
			min	max	avg	avg σ %	avg time	min	max	avg	avg σ %	avg time
STEP/RANDOM	100	27	27	27	27	0	128.20	27	27	27	0	57.70
	200	83	83	86	83.50	0.60	335.40	83	89	85.80	3.37	67.00
	300	168	168	172	169.50	0.89	305.10	172	183	176.30	4.94	75.50
	400	278	278	292	283.10	1.83	484.90	289	308	295.00	6.12	79.20
	500	420	420	426	421.90	0.45	463.20	431	453	440.50	4.88	82.30
	600	594	596	606	601.00	1.18	493.10	618	656	632.50	6.48	101.50
	700	815	821	875	855.70	4.99	821.10	818	862	846.30	3.48	91.90
	800	1133	1137	1194	1156.40	2.07	168.50	1172	1211	1187.00	4.77	102.50
	900	1644	1686	1896	1762.10	7.18	172.10	1651	1744	1701.70	3.51	90.20
	1000	2545	2735	2768	2749.90	8.05	331.70	2554	2853	2683.60	5.45	111.00
STEP/UNIT	20	20	20	21	20.10	0.50	93.10	20	20	20	0	51.80
	40	40	40	41	40.10	0.25	441.00	40	40	40	0	56.70
	60	59	59	60	59.30	0.51	840.10	59	59	59	0	68.70
	80	93	93	94	93.30	0.32	420.60	94	95	94.40	1.51	56.70
	100	132	132	132	132	0	406.30	133	135	134.20	1.67	81.60
	120	170	170	173	170.50	0.29	757.50	172	182	174.20	2.47	54.10
	140	215	217	224	220.70	2.65	1185.40	219	233	228.30	6.19	65.70
	160	296	343	378	355.10	19.97	686.90	298	351	331.10	11.86	81.30
	180	510	519	561	540.10	5.90	233.50	538	554	541.10	6.10	120.40
ALL/RANDOM	100	28	28	28	28	0	114.00	28	28	28	0	41.00
	200	84	84	84	84	0	153.30	85	90	88.30	5.12	57.70
	300	172	172	172	172	0	488.60	172	189	179.50	4.36	54.60
	400	287	**286**	287	**286.30**	-0.24	229.00	297	325	307.80	7.25	68.60
	500	435	435	435	435	0	237.60	451	486	464.50	6.78	50.40
	600	623	**622**	632	628.40	0.87	233.20	637	679	655.30	5.18	44.80
	700	867	869	900	886.90	2.30	320.20	874	908	889.60	2.61	41.70
	800	1236	1266	1340	1314.20	6.33	202.10	1254	1343	1277.50	3.36	38.30
	900	1950	1982	2202	2044.30	4.84	268.10	**1948**	2193	2118.90	8.66	46.60
	1000	3352	3535	3552	3545.50	5.77	389.40	3352	3423	3383.50	0.94	178.10
ALL/UNIT	20	20	20	20	20	0	42.50	20	20	20	0	47.30
	40	40	40	40	40	0	43.60	40	40	40	0	49.70
	60	60	60	60	60	0	474.20	60	60	60	0	63.70
	80	96	96	98	96.40	0.42	134.20	96	99	97.10	1.15	34.20
	100	139	**136**	**136**	**136**	-2.16	88.20	**136**	140	**137.80**	-0.86	31.60
	120	176	176	177	176.10	0.06	428.40	178	185	181.60	3.18	33.10
	140	227	234	253	245.90	8.33	338.90	234	239	235.70	3.83	35.50
	160	352	**351**	425	389.60	10.68	292.80	352	401	358.50	1.85	45.10
	180	639	639	639	639	0	474.50	639	641	639.20	0.03	379.20

Table A.2: Computational results of GA and SA for GSM

Problem (model/input)	K	B&B	GA					SA				
			min	max	avg	avg σ %	avg time	min	max	avg	avg σ %	avg time
STEP/RANDOM	100	18	18	18	18	0	64.90	18	18	18	0	53.50
	200	54	54	54	54	0	109.80	54	55	54.70	1.30	51.70
	300	107	**106**	**106**	**106**	-0.93	71.70	**106**	107	**106.50**	-0.47	58.50
	400	173	173	175	174.20	0.69	105.40	174	179	176.60	2.08	58.70
	500	260	260	264	260.40	0.15	119.60	263	278	269.60	3.69	57.70
	600	373	373	384	377.30	1.15	183.70	378	388	382.20	2.47	57.10
	700	519	519	523	520.40	0.27	202.70	521	543	531.70	2.45	56.80
	800	696	696	708	698.80	0.40	205.50	700	716	706.40	1.49	55.80
	900	919	931	943	935.90	1.84	218.30	919	953	936.90	1.95	67.70
	1000	1228	1264	1290	1273.60	3.71	162.80	1231	1296	1250.10	1.80	77.90
	1100	1540	1540	1544	1541.20	0.08	193.40	1540	1560	1546.00	0.39	140.20
STEP/UNIT	20	14	14	15	14.50	3.57	327.20	14	16	15.00	7.14	49.60
	40	31	31	33	31.80	2.58	105.40	31	32	31.90	2.90	54.50
	60	50	50	51	50.40	0.80	196.10	50	50	50	0	46.00
	80	69	69	69	69	0	49.00	69	69	69	0	43.90
	100	89	89	90	89.70	0.79	113.40	89	89	89	0	47.10
	120	108	109	110	109.30	1.20	438.00	109	110	109.90	1.76	54.20
	140	136	136	136	136	0	159.80	136	141	137.60	1.18	53.30
	160	175	175	182	180.30	3.03	265.90	175	183	177.10	1.20	44.20
	180	228	238	253	243.20	6.67	206.90	228	233	229.90	0.83	72.50
	200	293	**289**	**289**	**289**	-1.37	163.20	**289**	**290**	**289.60**	-1.16	161.80
ALL/RANDOM	100	19	**18**	**18**	**18**	-5.26	57.90	**18**	**18**	**18**	-5.26	37.90
	200	56	56	56	56	0	64.30	56	57	56.40	0.71	43.80
	300	110	110	110	110	0	79.40	111	111	111	0.91	49.50
	400	181	181	183	181.30	0.17	147.60	183	189	185.60	2.54	39.30
	500	280	**273**	**275**	**273.80**	-2.21	86.60	**274**	293	282.50	0.89	47.40
	600	392	392	400	397.80	1.48	151.90	394	403	398.40	1.63	46.20
	700	648	**554**	**560**	**557.00**	-14.04	114.30	**561**	**590**	**571.30**	-11.84	39.10
	800	745	745	751	749.40	0.59	126.90	747	768	757.90	1.73	40.10
	900	1028	**1013**	**1021**	**1020.20**	-0.76	131.10	**1008**	1063	1036.70	0.85	47.40
	1000	1332	1366	1410	1393.90	4.65	157.40	1351	1420	1404.60	5.45	73.70
	1100	1565	1565	1572	1567.50	0.16	171.70	1565	1603	1573.90	0.57	132.50
ALL/UNIT	20	14	14	14	14	0	232.70	14	15	14.80	5.71	41.30
	40	31	31	33	32.00	3.23	84.70	31	33	32.00	3.23	46.40
	60	51	51	51	51	0	24.30	51	51	51	0	36.30
	80	71	71	71	71	0	26.40	71	71	71	0	37.20
	100	91	91	91	91	0	37.00	91	91	91	0	40.90
	120	112	112	113	112.10	0.09	296.40	113	114	113.80	1.61	48.50
	140	144	144	144	144	0	96.40	144	147	145.40	0.97	48.20
	160	188	**187**	192	189.80	0.96	311.20	191	192	191.30	1.76	44.50
	180	240	251	252	251.30	4.71	160.80	240	250	247.50	3.12	63.50
	200	305	**291**	**291**	**291**	-4.59	127.10	**291**	**293**	**292.60**	-4.07	147.70

Appendix B

Additional results on ETSI TBR4 application

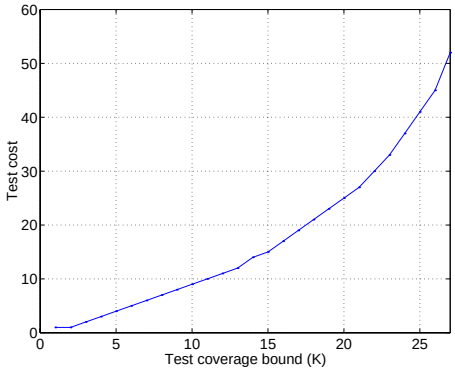

Figure B.1: Minimal cost problem, All or Nothing model, $\mathbf{c_1}$ cost vector

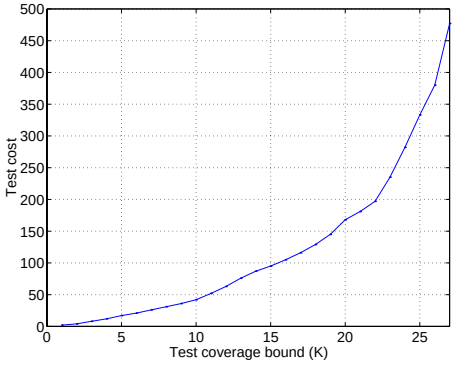

Figure B.2: Minimal cost problem, All or Nothing model, $\mathbf{c_2}$ cost vector

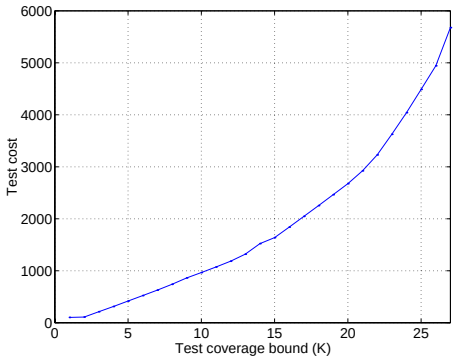

Figure B.3: Minimal cost problem, All or Nothing model, $\mathbf{c_3}$ cost vector

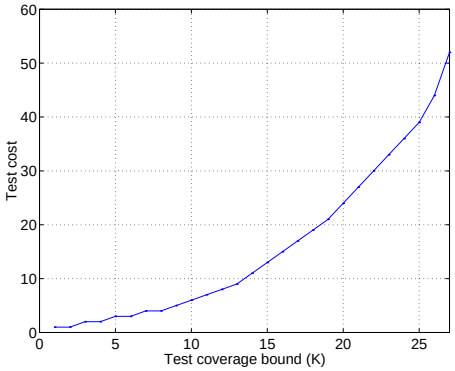

Figure B.4: Minimal cost problem, Linear model, $\mathbf{c_1}$ cost vector

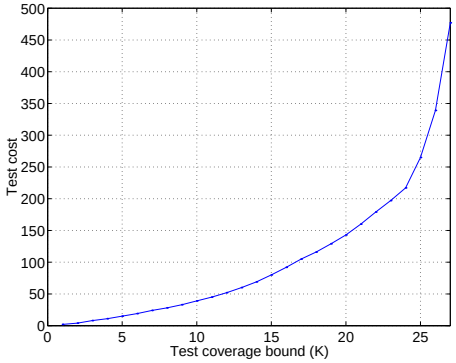

Figure B.5: Minimal cost problem, Linear model, $\mathbf{c_2}$ cost vector

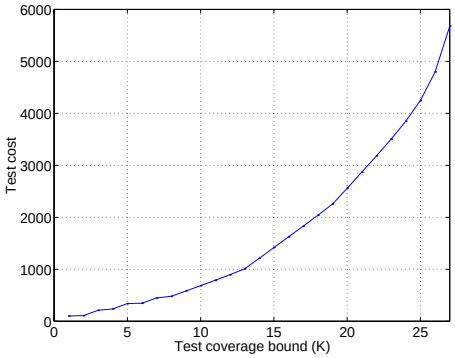

Figure B.6: Minimal cost problem, Linear model, $\mathbf{c_3}$ cost vector

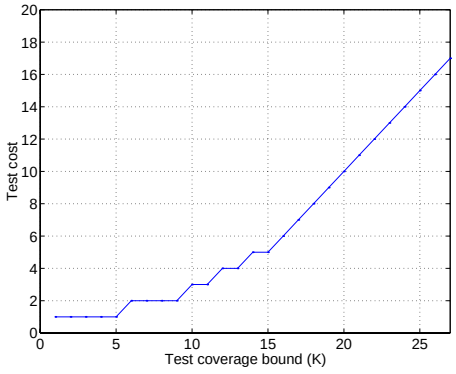

Figure B.7: Minimal cost problem, One is Enough model, $\mathbf{c_1}$ cost vector

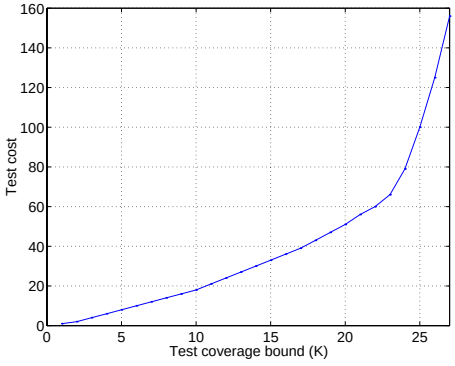

Figure B.8: Minimal cost problem, One is Enough model, $\mathbf{c_2}$ cost vector

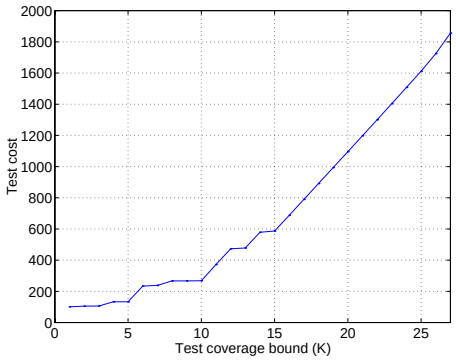

Figure B.9: Minimal cost problem, One is Enough model, $\mathbf{c_3}$ cost vector

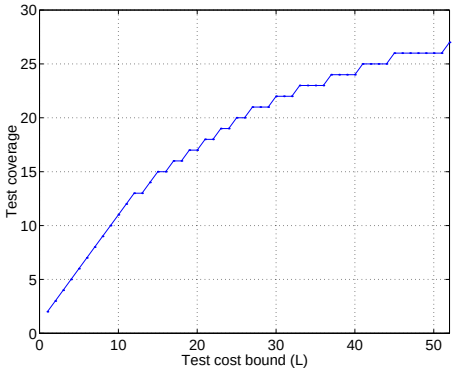

Figure B.10: Maximal coverage problem, All or Nothing model, $\mathbf{c_1}$ cost vector

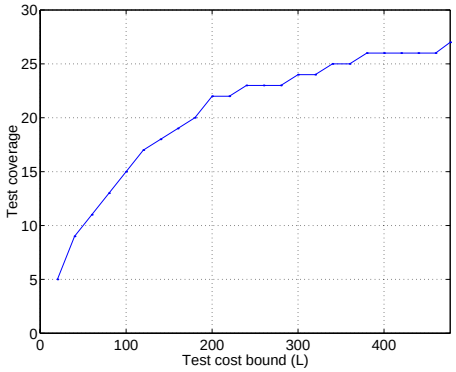

Figure B.11: Maximal coverage problem, All or Nothing model, $\mathbf{c_2}$ cost vector

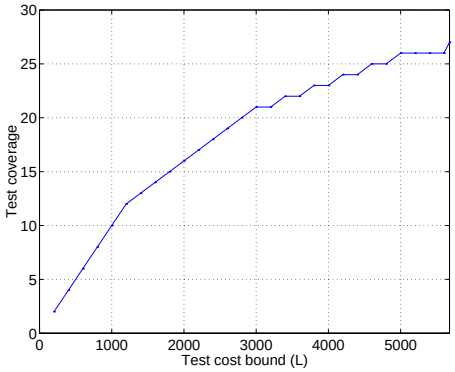

Figure B.12: Maximal coverage problem, All or Nothing model, $\mathbf{c_3}$ cost vector

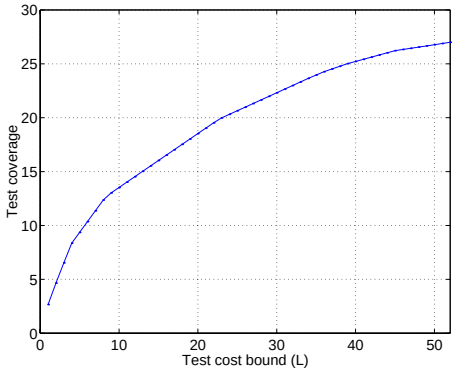

Figure B.13: Maximal coverage problem, Linear model, $\mathbf{c_1}$ cost vector

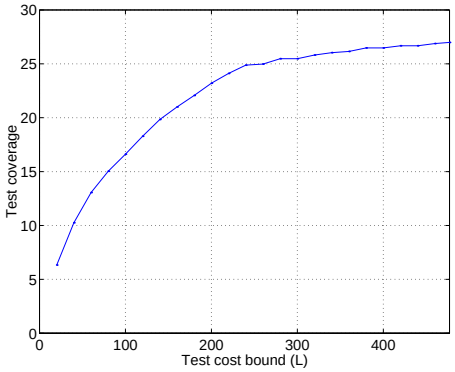

Figure B.14: Maximal coverage problem, Linear model, $\mathbf{c_2}$ cost vector

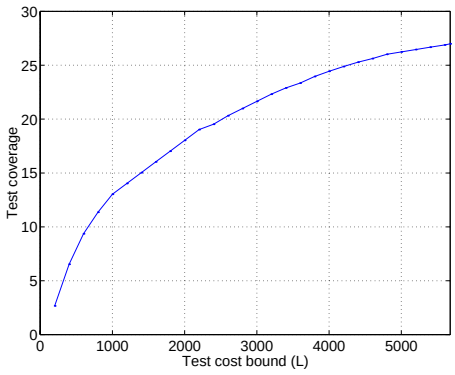

Figure B.15: Maximal coverage problem, Linear model, $\mathbf{c_3}$ cost vector

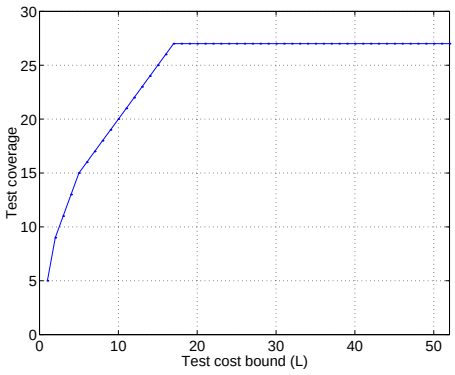

Figure B.16: Maximal coverage problem, One is Enough model, $\mathbf{c_1}$ cost vector

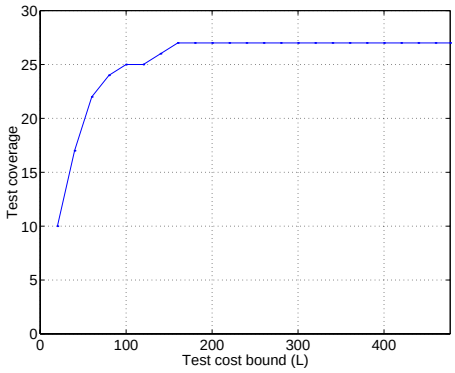

Figure B.17: Maximal coverage problem, One is Enough model, $\mathbf{c_2}$ cost vector

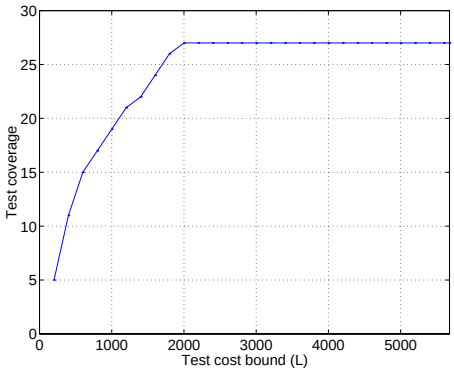

Figure B.18: Maximal coverage problem, One is Enough model, $\mathbf{c_3}$ cost vector

Appendix C

Additional results on function test application

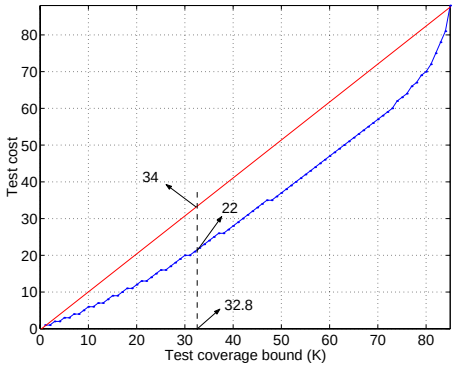

Figure C.1: Minimal cost problem, Linear model, $\mathbf{w_1}$ weight vector

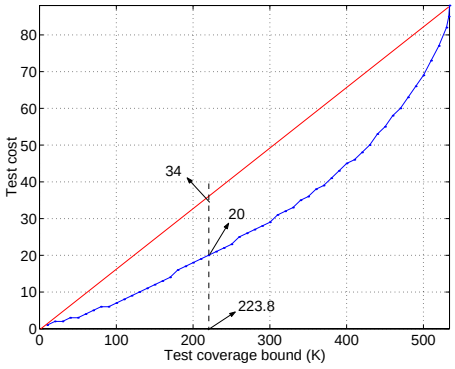

Figure C.2: Minimal cost problem, Linear model, $\mathbf{w_2}$ weight vector

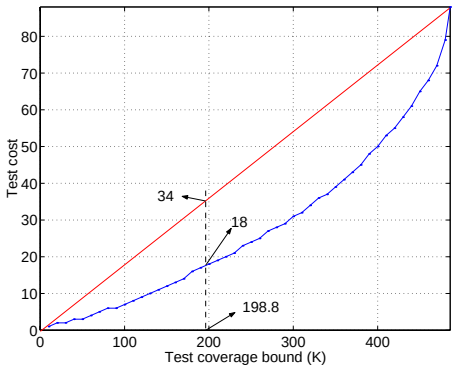

Figure C.3: Minimal cost problem, Linear model, $\mathbf{w_3}$ weight vector

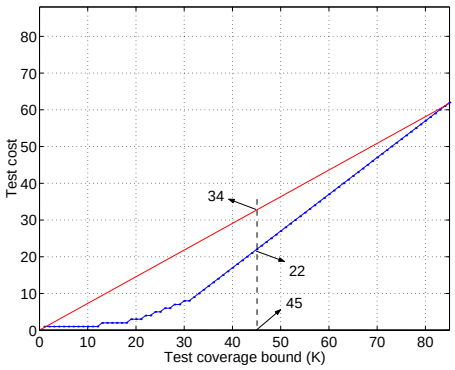

Figure C.4: Minimal cost problem, One is Enough model, $\mathbf{w_1}$ weight vector

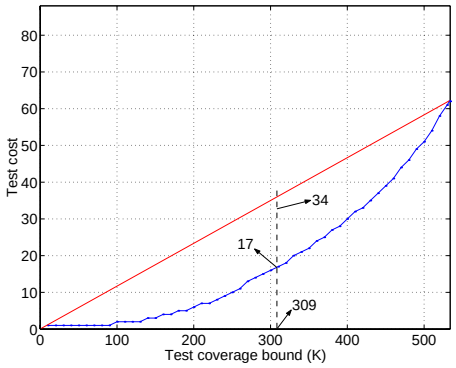

Figure C.5: Minimal cost problem, One is Enough model, $\mathbf{w_2}$ weight vector

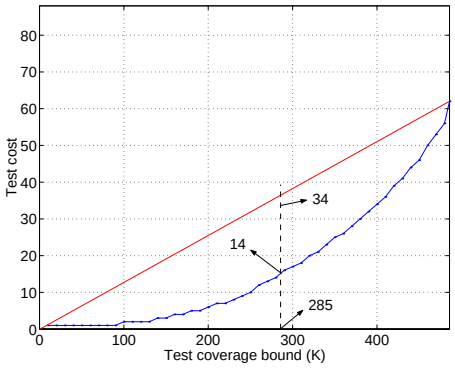

Figure C.6: Minimal cost problem, One is Enough model, $\mathbf{w_3}$ weight vector

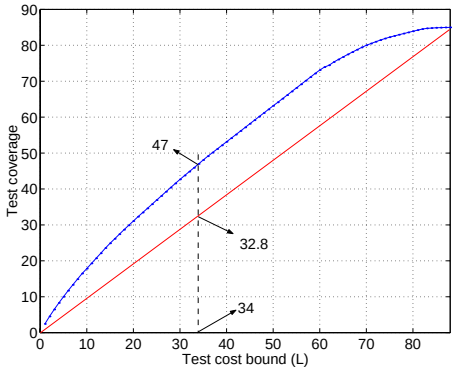

Figure C.7: Maximal coverage problem, Linear model, $\mathbf{w_1}$ weight vector

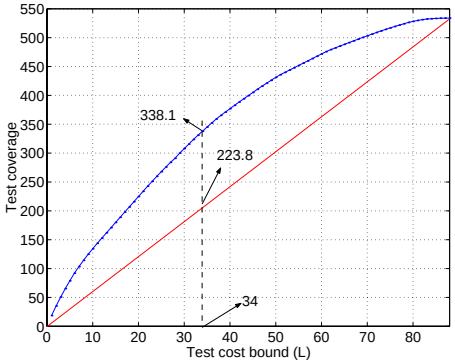

Figure C.8: Maximal coverage problem, Linear model, $\mathbf{w_2}$ weight vector

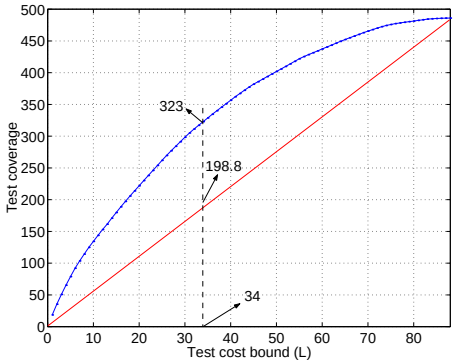

Figure C.9: Maximal coverage problem, Linear model, $\mathbf{w_3}$ weight vector

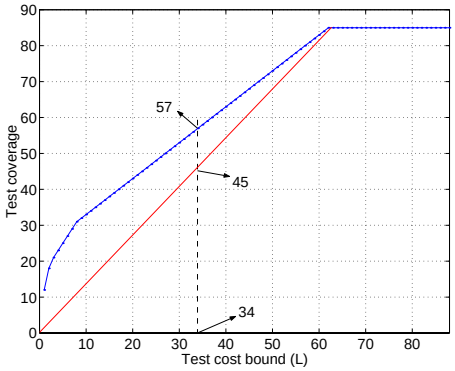

Figure C.10: Maximal coverage problem, One is Enough model, $\mathbf{w_1}$ weight vector

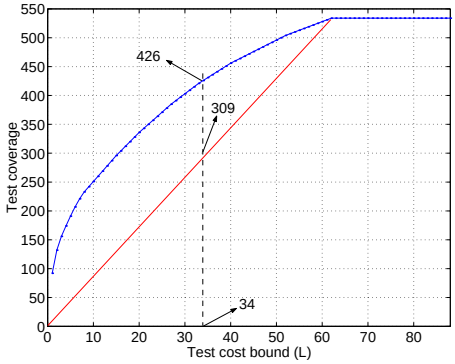

Figure C.11: Maximal coverage problem, One is Enough model, $\mathbf{w_2}$ weight vector

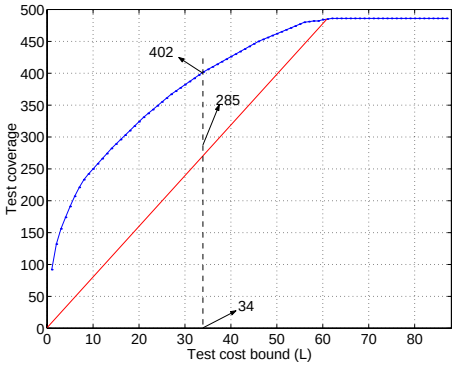

Figure C.12: Maximal coverage problem, One is Enough model, $\mathbf{w_3}$ weight vector

Abbreviations

ASP	Abstract Service Primitive
ATS	Abstract Test Suite
B&B	Branch and Bound
DS	Distinguishing Sequence
EFSM	Extended Finite State Machine
ETSI	European Telecommunications Standards Institute
FSM	Finite State Machine
GA	Genetic Algorithm
IP	Integer Programming
ILP	Integer Linear Programming
ISO	International Organization for Standardization
IUT	Implementation Under Test
OSI	Open Systems Interconnections
LTS	Labelled Transition Systems
PCTR	Protocol Conformance Test Report
PDU	Protocol Data Unit
PICS	Protocol Implementation Conformance Statement
PIXIT	Protocol Implementation eXtra Information for Testing
RTS	Reactive Tabu Search
SA	Simulated Annealing
TBR	Technical Basis for Regulation

TC	Test Case
TP	Test Purpose
TT	Transition Tour
TTCN	Tree and Tabular Combined Notation
UIO	Unique Input Output

Glossary of symbols

$\mathbb{R}_+$	non-negative real numbers
$\{a, b, c, \ldots\}$	the set containing elements a, b, c, ...; the order in which elements appear in a set is not important
$a \in A$	a is an element of set A
$\exists\, a \in A$	there exists an element a in set A
$\sum_{a \in A}$	sum for each a element of A
$\overset{def}{\Leftrightarrow}$	if and only if
$\overset{def}{=}$	defining equation
$x := y$	let x be equal to y
$x \longmapsto y$	y is image of x
i_k/o_l	input-output pair
$f : A \to B$	f is a function which maps each member of A to a member of B
$\{a \in A \mid P(a)\}$	the set containing all elements of A which satisfy predicate P. Sometimes $\{a \mid P(a)\}$ is used when the set A can be deduced from the context
$A \setminus B$	the set of elements which are in set A but not in set B

$A \subseteq B$	A is a subset of B, that is, all elements of A are also elements of B
$A \cap B$	the intersection of A and B, that is, the set containing all elements that are both in A and B
$A \times B$	the Cartesian product of A and B denoting the set of all ordered pairs (a, b) such that $a \in A$ and $b \in B$
$\|A\|$	number of elements in set A
$\|x\|$	the absolute value of x
$\lfloor x \rfloor$	the largest integer value not greater than x
$\lceil x \rceil$	the smallest integer value not less than x
$\lfloor x \rceil$	the closest integer to x
t_i $\underline{check}$ r_j	test case t_i is able to check requirement r_j
t_i $\underline{check}$ s_j	test case t_i is able to check subpurpose s_j
$purpcov(s_i, T)$	function of the number of test cases that are related to s_i
$\mathbf{a_i}$	n-dimensional characteristic row vector of T_{s_i}
b_i	the number of test cases related to subpurpose s_i
$c : TS \to \mathbb{R}_+$	cost function of the test cases
$\mathbf{c} = (c_1, c_2, \ldots, c_n)$	cost vector of the test cases
$c(t_i)$	cost of test t_i
$c(T)$	cost of test set T
$cov(T)$	coverage of test set T
$DATA = \{d_1, d_2, \ldots, d_q\}$	set of data elements

D_{t_i}	data elements sent or received in t_i
$f_i : \{0, 1, \ldots, \lvert T_{s_i} \rvert\} \to [0, 1]$	function defining the coverage ($i = 1, 2, \ldots, k$)
K	lower bound for the coverage
k	number of subpurposes
L	upper bound for the cost
n	number of test cases or test purposes
P_{t_i}	abstract test purpose for test case t_i
$REQ = \{r_1, r_2, \ldots, r_m\}$	set of conformance requirements
S_{t_i}	set of subpurposes that are checked by test case t_i
$SP = \{s_1, s_2, \ldots, s_k\}$	set of subpurposes
T	set of test cases
T_{s_i}	set of test cases that are able to check subpurpose s_i
$TP = \{p_1, p_2, \ldots, p_n\}$	set of test purposes
$TS = \{t_1, t_2, \ldots, t_n\}$	test suite: set of test cases
$\mathbf{x} = (x_1, x_2, \ldots, x_n)$	solution vector denoting the selected test cases
$\mathbf{x}_T$	characteristic vector of test set T
$w : SP \to \mathbb{R}_+$	weight function of the subpurposes
$\mathbf{w} = (w_1, w_2, \ldots, w_k)$	weight vector of the subpurposes
$w(s_i)$	weight of subpurpose s_i

Bibliography

[Aarts97] E. Aarts and J.K. Lenstra (Eds.), *Local Search in Combinatorial Optimization*, John Wiley & Sons, New York, 1997.

[Baum94] B. Baumgarten and A. Giessler, *OSI Conformance Testing Methodology and TTCN*, Elsevier, Amsterdam, 1994.

[Back93] T. Bäck, Optimal mutation rates in genetic search, in: S. Forrest (Ed.), *Proc. Fifth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, 2-9, 1993.

[Battiti94] R. Battiti and G. Tecchiolli, The Reactive Tabu Search, *ORSA Journal on Computing* **6(2)**, 126-140, 1994.

[Battiti96] R. Battiti, Reactive search: Toward self-tuning heuristics, in: V.J. Rayward-Smith (Ed.), *Modern Heuristic Search Methods*, John Wiley and Sons, Chichester, 61-83, 1996.

[Beasley96] J.E. Beasley and P.C. Chu, A Genetic Algorithm for the Set Covering Problem, *European Journal of Operational Research* **94**, 392-404, 1996.

[BGC] BGC Attendant ISDN-E Connection, Test description, *Ericsson internal document*, docno. 124/152 91-ANT 260 01/5 Uen, 1998.

[Brinksma87] T. Bolognesi and E. Brinksma, Introduction to the ISO specification language, LOTOS, *Computer Networks and ISDN Systems* **14**, 25-59, 1997.

[Bosik91] B.S. Bosik and M.Ü. Uyar, Finite state machine based formal methods in protocol conformance testing: from theory to implementation, *Computer Networks and ISDN Systems* **22**, 7-33, 1991.

[Bush90] M. Bush, K. Rasmussen and F. Wong, Conformance Testing Methodologies for OSI Protocols, *AT&T Technical Journal*, 84-100, January/February 1990.

[Caprara98] A. Caprara, M. Fischetti and P. Toth, Algorithms for the Set Covering Problem, *Research report* **OR-98-03**, University of Bologna, Italy, 1998. (http://www.or.deis.unibo.it/techrep.html)

[Cerny85] V. Cerny, A thermodynamical approach to the traveling salesman problem: An efficient simulated algorithm, *Journal of Optimization Theory and Applications* **45**, 41-51, 1985.

[Chow78] T.S. Chow, Testing Software Design Modeled by Finite-State Machines, *IEEE Transactions on Software Engineering*, Vol. SE-4, No. 3, 178-187, 1978.

[Chu97] P.C. Chu, *A Genetic Algorithm Approach for Combinatorial Optimisation Problems*, PhD Thesis, The Management School, Imperial College of Science, Technology and Medicine, London, 1997.

[CPLEX] CPLEX program documentation, version 3.0 1989-1994, CPLEX Optimization Inc.

[CsonEuro97] T. Csöndes, B. Kotnyek, A Mathematical Programming Method in Test Selection, in P. Milligan, P. Corr (Eds.), *New Frontiers of Information Technology,* IEEE Computer Society, Proceedings of the 23rd Euromicro Conference (Euromicro'97), 8-13, Budapest, Hungary, September 1–4, 1997.

[CsonKFKI97] T. Csöndes, Test Selection with Mathematical Programming, in K. Tarnay (Ed.), *Conformance Testing: Theory and Practice*, supported by COST 247 project and Hungarian Scientific Research Fund, KFKI-1997-05/M report, 24-32, 1997.

[CsonActa99] T. Csöndes, S. Dibuz, B. Kotnyek, Test Suite Reduction in Conformance Testing, *Acta Cybernetica,* Vol. 14, No. 2, 229-238, 1999.

[CsonIW99] T. Csöndes, B. Kotnyek, Automated Test Case Selection Based on Subpurposes, in Gy. Csopaki, S. Dibuz, K. Tarnay (Eds.), *Testing of Communicating Systems, Methods and Applications,* Kluwer Academic Publishers, 251-265, IFIP TC6 12^{th} International Workshop on Testing of Communicating Systems (IWTCS'99), Budapest, Hungary, September 1–3, 1999.

[Csondes00] T. Csöndes, S. Dibuz. P. Krémer, Experiments on IPv6 Testing, in H. Ural, R.L. Probert, G.V. Bochmann (Eds.), *Testing of Communicating Systems, Tools and Techniques,* Kluwer Academic Publishers, 113-126, IFIP TC6/WG6.1 13^{th} International Conference on Testing of Communicating Systems (TestCom 2000), Ottawa, Canada, August 29–September 1, 2000.

[CsonFAT01] T. Csöndes, B. Kotnyek, A Formal Approach to Practical Test Selection, in E. Brinksma, J. Tretmans (Eds.), *Formal Approaches to Testing of Software (FATES'01),* Proceedings, A Satelit Workshop of CONCUR'01, Aalborg, Denmark, August 25, 2001.

[CsonEJ01] T. Csöndes, B. Kotnyek, J. Z. Szabó, Application of Heuristic Methods for Conformance Test Selection, *European Journal of Operational Research,* accepted paper, August 3, 2001.

[Dahbura90] A.T. Dahbura, K.K. Sabnani and M.Ü. Uyar, Algorithmic Generation of Protocol Conformance Tests, *AT&T Technical Journal*, 101-118, January/February 1990.

[Dibuz97] S. Dibuz, T. Csöndes, B. Kotnyek, Conformance Testing of Communication Protocols, in K. Boyanov (Ed.), *Network Information Processing Systems,* Proceedings of the IFIP TC6 International Symposium, 48-58, Sofia, Bulgaria, October 14–16, 1997.

[ETR266] ETSI ETR 266; Methods for Testing and Specification (MTS); Test Purpose style guide, 1996.

[ETS374] ETSI ETS 300 374-4, Core Intelligent Network Application Protocol (INAP), Abstract Test Suite (ATS)

[ETS403] ETSI draft prETS 300 403-7: Integrated Services Digital Network (ISDN); Digital Subscriber Signalling System No. one (DSS1) protocol; Signalling

network layer for circuit-mode basic call control; Part 7: Abstract Test Suite (ATS) and partial Protocol Implementation eXtra Information for Testing (PIXIT) proforma specification for the network, 1998.

[ETS607] ETSI ETS 300 607-3 ed. 10 (1999-04): Digital cellular telecommunications systems (Phase 2); Mobile Station (MS) conformance specification; Part 3: Layer 3 (L3) Abstract Test Suite (ATS) (GSM 11.10-3 version 4.24.1), 1999.

[TTCN-3] ETSI DES/MTS-00063-1 v1.0.10; Methods for Testing and Specification (MTS); The Tree and Tabular Combined Notation version 3; TTCN-3: Core Language, 2000.

[TBR4] ETSI Draft prTBR4 Integrated Services Digital Network (ISDN); Attachment requirements for terminal equipment to connect to an ISDN using ISDN primary rate access, 1994.

[ISO8807] ISO/IEC 8807, Information Processing Systems - Open Systems Interconnection - LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour, 1989.

[ISO9646] ITU-T Recommendation X.290-X.296 - ISO/IEC 9646, Information Technology - Open Systems Interconnection - Conformance Testing Methodology and Framework, 1994.

[FMCT] ITU-T Recommendation Z.500, Framework on Formal Methods in Conformance Testing, 1997.

[Frank97] J.D. Frank, *Local Search for NP-hard Problems*, PhD Thesis, Computer Science in the Office of Graduate Studies, University of California, Davis, 1997.

[Garey79] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.

[Glover86] F. Glover, Future paths for integer programming and links to artificial intelligence, *Computers & Operations Research* **5**, 533-549, 1986.

[Glover89] F. Glover, Tabu Search - part I., *ORSA Journal on Computing* **1 (3)**, 190-260, 1989.

[Glover90] F. Glover, Tabu Search - part II., *ORSA Journal on Computing* **2 (1)**, 4-32, 1990.

[Goldberg89] D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA, 1989.

[Gonenc70] G. Gönenc, A method for the design of fault detection experiments, *IEEE Transactions on Computers*, Short Notes, Vol. 19, No. 6, 551-558, 1970.

[Heijink94] R.J. Heijink, FAITH, a General Purpose Test System for ISDN, *Computer Networks and ISDN Systems* **26**, 1581-1593, 1994.

[Holland75] J.H. Holland, *Adaption in Natural and Artificial Systems*, MIT Press, Cambridge, MA, 1975.

[Holzmann91] G.J. Holzmann, *Design and Validation of Computer Protocols*, Prentice-Hall, Englewood Cliffs, New Jersey, 1991.

[Johnson89] D.S. Johnson, C.R. Aragon, L.A. McGeoch and C. Schevon, Optimization by Simulated Annealing: An Experimental Evaluation; Part I, Graph Partitioning, *Operations Research*, Vol. 37, No. 6, 865-892, 1989.

[Johnson91] D.S. Johnson, C.R. Aragon, L.A. McGeoch and C.Schevon, Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring and Number Partitioning, *Operations Research*, Vol. 39, No. 3, 378-406, 1991.

[Kirkpatrick83] S. Kirkpatrick, C.D. Gelatt Jr. and M.P. Vecchi, Optimization by simulated annealing, *Science* **220**, 671-680, 1983.

[Knuth73] D.E. Knuth, *The Art of Computer Programming, Vol. 3: Sorting and searching*, Addison-Wesley, Reading, MA, 1973.

[Kohavi78] Z. Kohavi, *Switching and Finite Automata Theory*, McGraw-Hill, New York, 1978.

[Milner80] R. Milner, *A Calculus of Communicating Systems*, Lecture Notes in Computer Science 92, Springer Verlag, New York, 1980.

[Naito81] S. Naito and M. Tsunoyama, Fault Detection for Sequential Machines by transition tours, in *Proc. 11th IEEE Fault Tolerant Comput. Symp.*, IEEE Computer Soc. Press, 238-243, 1981.

[Nemhauser98] G. L. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization*, Wiley, New York, 1998.

[OSLMIP] See at http://www6.software.ibm.com/es/oslv2/features/mip.htm, 2001.

[Pirlot96] M. Pirlot, General Local Search Methods, *European Journal of Operational Research* **92**, 493-511, 1996.

[Probert92] R.L. Probert and O. Monkewich, TTCN: The International Notation for Specifying Tests of Communications Systems, *Computer Networks and ISDN Systems* **23**, 417-438, 1992.

[Sabnani88] K. Sabnani and A. Dahbura, A Protocol Test Generation Procedure, *Computer Networks and ISDN Systems* **15**, 285-297, 1988.

[Sarikaya82] B. Sarikaya and G. v. Bochmann, Some Experience with Test Sequence Generation for Protocols, in C. Sunshine (Ed.), *Protocol Specification, Testing and Verification*, North Holland, 555-567, 1982.

[Sarikaya92] B. Sarikaya and A. Wiles, Standard Conformance Test Specification Language TTCN, *Computer Standards & Interfaces* **14**, 117-144, 1992.

[Sidhu89] D.P. Sidhu and T.K. Leung, Formal Methods for Protocol Testing: A Detailed Study, *IEEE Transactions on Software Engineering*, Vol. 15, No. 4, 413-426, 1989.

[Sun97] X. Sun, Y. Shen, C. Feng and F. Lombardi, *Protocol Conformance Testing Using Unique Input/Output Sequences*, World Scientific, Singapore, 1997.

[Tan97] Q. Tan, *On Conformance Testing of Systems Communicating by Rendezvous*, PhD thesis, Université de Montréal, Canada, 1997.

[Tarnay91] K. Tarnay, *Protocol Specification and Testing*, Akadémiai Kiadó, Budapest, 1991.

[Tretmans92] J. Tretmans, *A Formal Approach to Conformance Testing*, PhD thesis, University of Twente, Hengelo, The Netherlands, 1992.

[Uyar98] M.Ü. Uyar, Dual-state Augmentation for Minimizing Conformance Test Costs, *Computer Networks and ISDN Systems* **30**, 1277-1294, 1998.

[Vuong92] S.T. Vuong and J. Alilovic-Curgus, On Test Coverage Metrics for Communication Protocols in J. Kroon, R.J. Heijink, E. Brinksma (Eds.), *Protocol Test Systems*, IV, Elsevier, 31-45, 1992.

[Vuong97] S.T. Vuong, J. Zhu and J. Alilovic-Curgus, Sensitivity Analysis of the Metric Based Test Selection in M. Kim, S. Kang, K. Hong (Eds.) *Testing of Communicating Systems*, Vol. 10, Chapman & Hall, 1997.

[Wolsey98] L.A. Wolsey, *Integer Programming*, Wiley, 1998.

[Zhu97] J. Zhu and S.T. Vuong, Generalized Metric Based Test Selection and Coverage Measure for Communication Protocols in T. Mizuno, N. Shiratori, T. Higashino, A. Togashi (Eds.), *Formal Description Techniques and Protocol Specification, Testing and Verification*, Chapman & Hall, 299-314, 1997.