Acta Cybernetica **20** (2012) 459–481.

Model-Driven Diagnostics of Underperforming Communicating Systems*

Levente Erős† and Tibor Csöndes‡

Abstract

This paper proposes methods for improving the performance of a communicating system that has failed its performance test. The proposed methods extend our earlier published model-driven performance testing method, which automatically determines whether the tested system is able to serve the specified number of requests within a second in worst case while serving a specified number of users simultaneously. The underperformance diagnostic methods presented in this paper are given as an input the formal performance model representing the system under test, which was built up by our performance testing method in the performance testing phase. The presented methods aim at improving the performance of the system under test to the desired level at minimal cost. One of the methods presented in this paper is a binary linear program, while the other is a heuristic method which, according to our simulation results, performs efficiently.

Keywords: performance testing, performance diagnostics, complexity theory, optimization, approximation algorithms

1 Introduction

Testing is the last phase of the development of a system implementing a communication protocol. The kinds of tests run on a communicating system can be several, but two of the most important kinds of tests are conformance tests and performance tests. When performing a black-box test on a communicating system, the test enviroment (or tester) does not know anything about the internal structure of the system under test (SUT) and can only investigate the SUT through its responses (outputs) given for different requests (inputs). Black-box conformance testing examines whether the SUT implements the communication protocol that it should

*This paper has been (partially) supported by HSNLab, Budapest University of Technology and Economics, http://www.hsnlab.hu.

†Department of Telecommunications and Media Informatics, Budapest University of Technology and Economics, H-1117. Magyar tudósok krt. 2. Budapest, Hungary, E-mail: eros@tmit.bme.hu

‡Ericsson Hungary, H-1117. Irinyi József u. 4-20. Budapest, Hungary, E-mail: tibor.csondes@ericsson.com

DOI: 10.14232/actacyb.20.4.2012.1

implement, according to its conformance requirements. During a conformance test, the SUT is compared to a Finite State Machine (FSM) based formal model which is created according to the written conformance specifications of the SUT [1]. A performance test can be executed on the SUT once it has passed its conformance test that is, once it has been found to correspond to the FSM model representing its conformance requirements. During a performance test, the tester tries to determine whether the SUT meets its different kinds of performance requirements. One possible performance requirement is the number of request messages that the SUT has to be able to process within a second in worst case while serving a specified number of users in parallel.

If the SUT fails the performance test run on it, its performance has to be improved. If the SUT fails the performance test because the number of requests it can serve within a second in worst case while serving a specified number of users simultaneously is lower than the number of messages specified as a requirement, the number of request messages it is able to serve within a second has to be increased. In this paper, we present underperformance diagnostic methods that, given such a system, attempt to determine how to improve its performance to the desired level at minimal cost.

The rest of the paper goes as follows: In Section 2, we give a summary on the related work in the subject. In Section 3, we briefly review our performance testing method which is the basis of the presented underperformance diagnostic methods. In Section 4, we deal with the worst-case underperformance diagnostics problem. Within Section 4, in Subsection 4.1 we define the worst-case underperformance diagnostics problem, in Subsection 4.2 we prove that the problem is NP-complete, in Subsection 4.3 we formulate the problem as a binary linear program, while in Section 4.4 we give a heuristic solution for the problem. In Section 5, we evaluate the efficiency of our heuristics by analyzing simulation results. We close the paper with a summary in Section 6, and a few words on our future work in Section 7.

2 Related Work

Conformance testing of communicating systems has a well-developed scientific background [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15], and many papers have been written in the field of modeling the performance of a system which implements a communicating protocol as well. In [16], Kemper et al. present a performance model for communicating systems, based on SDL (Specification and Description Language) [17]. Youness et al. [18] use a stochastic Petri net [19] while El-Karaksy et al. [21] use a timed Petri net [20] for modeling the performance of a communication protocol. Marsan et al. model the performance of CSMA/CD bus LANs by a timed Petri net based model [22]. These papers also *verify* the models presented that is, they analytically prove that their models correspond to the specifications however, neither of them examine whether a *physical system* corresponds to its performance requirements that is, neither of these papers deal with the *validation* of communicating systems, which has been our goal during our earlier research work.

There are a few papers in the literature dealing with the performance validation of communicating systems. Schieferdecker et al. propose PerfTTCN [23]. PerfTTCN is a language extension of TTCN-2 (Tree and Tabular Combined Notation ver. 2) [24]. It introduces new language elements for describing performance testing environments and for conducting performance tests. As a language extension of TTCN-3 (Testing and Test Control Notation ver. 3) [25], Grabowski et al. introduce TimedTTCN [26], which tests the response delays of the SUT. Mingwei et al. [27] extend concurrent TTCN [28] (which is a version of TTCN-2) to be able to test the performance of communication protocols. These three papers introduce useful tools and language extensions for TTCN but unfortunately, neither of them deals with how to generate the performance test itself, based on the performance requirements of the SUT.

To sum up the above, during our earlier work, we were unable to find any automatic, black-box methods in the literature that validate the performance of communicating systems. Thus, we have proposed an automatic model-driven performance testing method. Our performance testing method is given as an input the number of requests the SUT has to be able to serve in worst case while serving a given number of users in parallel. The method automatically determines whether the SUT meets this performance requirement [29]. The underperformance diagnostic methods presented in this paper are the extensions of this earlier published performance testing method, which is briefly discussed in Section 3.

3 A Model-Driven Performance Testing Method for Communicating Systems

In this section, we briefly introduce an automatic performance testing method. The method is given as an input CWR_{usr}, which is the worst-case number of requests the SUT has to process within a second (worst-case number of messages per second), while serving *usr* users simultaneously. The performance testing method automatically determines, whether the SUT meets this performance requirement. More precisely, our performance testing method calculates CW_{usr}, which is the number of messages the SUT is able to process within a second while serving *usr* users. The method surveyed in this section is presented in [29] in more detail.

For the performance testing method presented in this section, we have created a performance model, the so-called Timed Communicating Finite Multistate Machine (TCFMM), which is capable of modelling the above mentioned two performance requirements of a communicating system. During testing, our performance testing method creates a TCFMM model representing the SUT. This model is created using the information known about the SUT prior to testing, and using measurements performed on the SUT during the testing phase. From the so created performance model, the method derives CW_{usr} in an analytical way. The Timed Communicating Finite Multistate Machine can be described as follows:

$$TCFMM = (S, T, I, O, U, s_0)$$

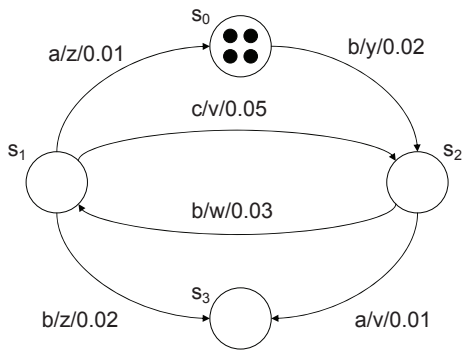

Figure 1: The TCFMM model

In the above definition, S is the set of states with $s_0 \in S$ being the initial state of the TCFMM, T is the set of transitions, I is the set of inputs, O is the set of outputs, and U is the set of tokens in the TCFMM. Each transition $t_i \in T$ is a quintuple $(s_{from_i}, s_{to_i}, i_i, o_i, d_i)$, where $s_{from_i} \in S$ is the originating state, $s_{to_i} \in S$ is the destination state, $i_i \in I$ is the input, $o_i \in O$ is the output, and d_i is the delay of the transition. Each token $u_j \in U$ represents a protocol instance run by the SUT. Each u_j has a current state $s_{current_j}$, which is the current state of that protocol instance of the SUT, which communicates with user j. The transitions work as follows. Let us assume that $s_{current_j} = s_{from_i}$. If user j sends i_i to the system represented by the TCFMM, token u_j is removed from s_{from_i} and placed to s_{to_i} and o_i is sent to the user in response. The time elapsed between the user sending i_i to and receiving o_i from the system represented by the TCFMM is d_i. In the beginning, for each $u_j \in U$, $s_{current_j} = s_0$. Figure 1 shows a TCFMM. The transition parameters are written on each transition in the form *input/output/delay*. All the tokens of the TCFMM reside in s_0.

As the first step, our performance testing method creates the structure of the TCFMM representing the SUT. In the rest of the paper, by the structure of the TCFMM, we mean the complete TCFMM without its transition delays d_i. In the testing phase, the tester measures all transition delays of the SUT and completes this TCFMM model. During testing, the structure of the TCFMM is used for tracing the state of the SUT.

The structure of the TCFMM is based on the FSM model to which the SUT corresponds, according to its conformance test. Each state of the TCFMM corresponds to a state of the FSM, while each of its transitions correnspond to a transition in the FSM. The input and output value of a transition in the TCFMM equals the input and output value of the corresponding transition in the FSM, respectively. Two states in the TCFMM are connected by a transition exactly if the corresponding states are connected by the corresponding transition in the FSM. The originating and destination states of each transition in the TCFMM are the originating and

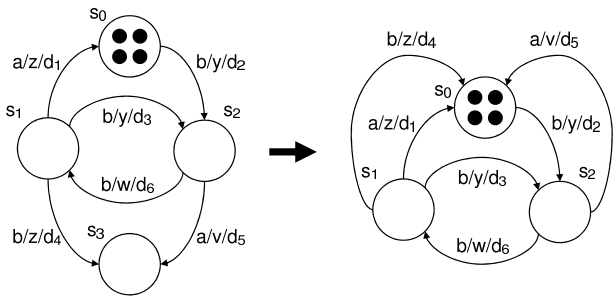

Figure 2: Redirecting transitions from terminating states

destination states of the corresponding transition in the FSM, respectively. After creating the states and transitions, *usr* tokens are placed to s_0 in the TCFMM. Placing these tokens to state s_0 in the TCFMM means that during the performance measurement, the tester will emulate the maximal number of users the SUT has to be able to handle. During testing, moving token u_j along transition t_i from state s_{from_i} to state s_{to_i} corresponds to the tester sending input i_i to the SUT and then waiting to receive output o_i from the SUT, in the name of user j.

To complete the structure of the TCFMM, all the transitions leading to sink states (i.e. states that have no outgoing transitions) have to be redirected to s_0, and the sink states have to be elliminated. The reason for this modification is that if there were sink states in the TCFMM used for conducting the performance test and a token u_i reached one of these sink states, then the tester would not be able to send any request messages (inputs) to the SUT in the name of user i that is, the effective number of users the SUT has to serve would be decreased by one due to this stuck token u_i. In other words, token u_i would go inactive. If however, the transition leading u_i to this sink state is redirected to s_0, every time a token goes through this transition, it reappears at s_0 instead of going inactive. This is identical to the situation when for each user that sends its last request to the SUT and goes inactive, a new user appears. With this modification, the number of users that the SUT has to serve simultaneously is *usr* for the whole duration of the test. Figure 2 shows two TCFMMs. The TCFMM in the right side of the figure is the TCFMM in the left side of the figure, after its transitions leading to the only sink state s_3 got redirected to s_0.

After creating the structure of the TCFMM, the tester measures the yet unknown transition delay values d_i on the SUT. During testing, each user emulated by the tester sends one request after another to the SUT. Thus, upon receiving a response from the SUT, the user sends the next request right away, and this way the SUT is continuously stressed by *usr* requests from *usr* users. Once each d_i is known, the TCFMM is a complete performance model of the SUT. Based on this TCFMM, CW_{usr} can be calculated.

Before going on with calculating CW_{usr}, let us define what it means that a

system is able to process CWR_{usr} messages per second in worst case.

Definition 1. *Let F_t^s denote the number of state transitions of the SUT measured for time length t while the SUT is fed by s. Then the SUT is said to be able to process CWR_{usr} messages per second if for an arbitrary infinite input sequence s,*

$$\lim_{t\to\infty} \frac{F_t^s}{t} \geq CWR_{usr} \tag{1}$$

The above fraction is the reciprocal of the average amount of time needed to process one input message of s. Since the amount of time needed to process any input sequence of s equals a transition delay which takes its value from a finite set, this average delay does have a limit and thus, the limit in the above formula exists too.

According to the above definition, a system is said to be able to process CWR_{usr} messages per second in worst case if it processes at least CWR_{usr} messages per second when induced by an *arbitrary* and infinite sequence of inputs, measured for a relatively long (optimally infinite) period of time.

In the following, c_i represents a cycle of transitions in the TCFMM, while $|c_i|$ represents its length. The following is a sufficient and necessary requirement of a system that processes CWR_{usr} messages per second in worst case: A system is able to process CWR_{usr} messages per second in worst case if and only if, for each c_i of the TCFMM of the system:

$$\sum_{t_j \in c_i} d_j \leq \frac{|c_i|}{CWR_{usr}} \tag{2}$$

As a consequence of the above, the number of messages the SUT is able to process within a second in worst case can be calculated as follows, where C is the set of all transition cycles in the TCFMM:

$$CW_{usr} = \min_{c_i \in C} \{ \frac{|c_i|}{\sum\limits_{t_j \in c_i} d_j} \} \tag{3}$$

4 The Worst-Case Underperformance Diagnostics Problem

After the above introduction, we are going to show how to increase the performance of a communicating system for which, $CWR_{usr} > CW_{usr}$ (in other words, the system is unable to process CWR_{usr} messages per second in worst case).

Increasing CW_{usr} is achieved by reducing the transition delays of the SUT. We are going to assume that transition delays are not reducible by arbitrary amounts. Moreover, each transition delay d_i is reducible by amounts $\frac{d_i}{Gr}, 2\frac{d_i}{Gr}, \ldots, (Gr-1)\frac{d_i}{Gr}$, where Gr (the so-called granularity) is a positive integer. Each transition delay reduction has a cost. The objective of the methods presented in this section is

to correct (some of the) transition delays of the SUT, so that at the end of the correction $CWR_{usr} \leq CW_{usr}$ and to carry out this correction at minimal cost.

4.1 Definition of the Worst-Case Underperformance Diagnostics Problem

The worst-case underperformance diagnostics problem is defined as follows:

Given are the set $T = \{t_i\}$ of transitions, and the set $C = \{C_i\}$ of cycles. Each cycle $C_i = \{t_j\}$ is a set of transitions, and each transition t_j has a delay value d_j. Given are a positive integer Gr, a positive number CWR_{usr}, a positive number K, and a variable $0 < q_i \leq 1$ assigned to each transition t_i. Given is furthermore, a monotonic decreasing function $Cost(x)$ for which $Cost : (0, 1] \to \mathbb{R}^+$, $Cost(1) = 0$ measured by discrete equidistant points of the domain. The question to be answered is as follows: Is it possible to choose the value of each q_i so that $q_i = \frac{n_i}{Gr}$, where $0 < n_i \leq Gr$ is an integer and the following inequalities are true?

$$\forall c_i \in C : \sum_{j:t_j \in c_i} d_j q_j \leq \frac{|c_i|}{CWR_{usr}} \tag{4}$$

$$\sum_{i:t_i \in T} Cost(q_i) \leq K \tag{5}$$

To further explain the above, q_i is a factor representing the reduction of d_i (a correction factor from now on). The reduced delay of each $t_i \in T$ is $d_i q_i$. $\sum_{i:t_i \in T} Cost(q_i)$ is the total cost of delay reduction. $Cost(1) = 0$, because if $q_i = 1$, the delay of t_i is not reduced, and so the delay reduction does not cost anything. Finally, K is an upper bound for the cost of correcting the delays of all transitions. Formula 4 corresponds to Formula 2 thus, it expresses that after the performance correction, $CW_{usr} \geq CWR_{usr}$. Formula 5 requires the cost of the correction to be under K.

4.2 Complexity of the Worst-case Underperformance Diagnostics Problem

In this subsection, we are going to prove the NP-completeness of the worst-case underperformance diagnostics problem by reducing an arbitrary instance of the NP-complete knapsack problem to an instance of the worst-case underperformance diagnostics problem, using the Karp reduction.[30]

Proof. Before beginning the proof, let us redefine the worst-case underperformance diagnostics problem using the attributes of the first definition:

Given are the set $T = \{t_j\}$ of transitions, and the set $C = \{C_i\}$ of cycles. Each cycle $C_i = \{t_j\}$ is a set of transitions. Each transition $t_j = \{(d_{jk}, c_{jk})\}$ is a set of delay-cost pairs, where $d_{jk} = d_j \frac{k}{Gr}$, $c_{jk} = Cost(\frac{k}{Gr})$, and $1 \leq k \leq Gr$ is an integer $(\forall (t_j \in T) : |t_j| = Gr)$. The question to be answered is as follows: Is it possible

to choose exactly one delay-cost pair $(\hat{d_j}, \hat{c_j}) \in t_j$ from each transition t_j so that $\forall(i : C_i \in C) : \sum_{j:t_j \in C_i} \hat{d_j} \leq \frac{|C_i|}{CWR_{usr}}$ and $\sum_{j:t_j \in T} \hat{c_j} \leq K$? To further explain the above, for each transition t_i, $d_{i|t_i|}$ is the measured delay d_i of the transition and $c_{i|t_i|} = Cost(\frac{|t_i|}{Gr}) = Cost(\frac{Gr}{Gr}) = Cost(1) = 0$.

A set $\hat{T}$ of the chosen $(\hat{d_j}, \hat{c_j})$ pairs is an appropriate witness, since given this set (containing $|T|$ elements), checking whether the elements of $\hat{T}$ give an appropriate solution can be done by summing up the $\hat{d_j}$ values and checking whether the sum is lower than or equal to $\frac{|C_i|}{CWR_{usr}}$, and by summing up the $\hat{c_j}$ values and checking whether their sum is lower than or equal to K. This operation can be carried out in $O(|T||C|)$ time that is, in polynomial time. Thus, the worst-case underperformance diagnostics problem is in NP.

Now, we have to reduce an arbitrary instance of the knapsack problem to an instance of the worst-case underperformance diagnostics problem. The knapsack problem is defined as follows:

Given are a set G, for all of its elements g_j a $v(g_j)$ and a $w(g_j)$ value and positive integers V and W. The question to be answered is as follows: Is there a subset $G' \subseteq G$ such that the following inequalities are true?

$$\sum_{g_j \in G'} w(g_j) \leq W \tag{6}$$

$$\sum_{g_j \in G'} v(g_j) \geq V \tag{7}$$

Let us now take this definition of the knapsack problem and reduce it to an instance of the worst-case underperformance diagnostics problem. First of all, to each $g_j \in G$ of the knapsack problem, a transition t_j is assigned, such that $t_j = \{(d_{j1}, c_{j1}), (d_{j2}, c_{j2})\}$, and $\bigcup_j t_j = T$. The variables of the resulting worst-case underperformance diagnostics problem are as follows:

$$\begin{array}{rcc} d_{j1} & = & v(g_j) \\ c_{j1} & = & w(g_j) \\ d_{j2} & = & 2v(g_j) \\ c_{j2} & = & 0 \\ C & = & \{C_1\} \\ C_1 & = & T \\ CWR_{usr} & = & \frac{|G|}{2\sum_{g_i \in G} v(g_i) - V} \\ K & = & W \end{array}$$

According to the assignments above, in the resulting graph there will be exactly one cycle containing all the transitions. Furthermore, each transition t_j will have two delay-cost pairs. Choosing pair (d_{j1}, c_{j1}) in the worst-case underperformance diagnostics problem corresponds to including g_j in G' in the knapsack problem, while choosing pair (d_{j2}, c_{j2}) corresponds to not including g_j in G'.

Now we have to show that the knapsack problem is solvable if and only if the corresponding worst-case underperformance diagnostics problem is solvable.

Let us assume that the above defined worst-case underperformance diagnostics problem is solvable. This means that for each $t_j \in T$ there is a $(\hat{d_j}, \hat{c_j}) \in t_j$ pair such that the following inequalities are true:

$$\sum_{j:t_j \in C_1} \hat{d_j} \leq \frac{|C_1|}{CWR_{usr}} \tag{8}$$

$$\sum_{j:t_j \in T} \hat{c_j} \leq K \tag{9}$$

Using the assignments defined earlier in this proof, Inequality 8 can be transformed as follows:

$$\begin{aligned} 2 \sum_{g_i \in G} v(g_i) - \sum_{g_i \in G'} v(g_i) \leq \\ \leq \frac{|C_1|}{\frac{|G|}{2 \sum\limits_{g_i \in G} v(g_i) - V}} \end{aligned} \tag{10}$$

The reason for transforming the left side of Inequality 8 as above is the following:

According to the assignments defined earlier in the proof, $d_{j1} = v(g_j) = 2v(g_j) - v(g_j)$. Thus, each $\hat{d_j}$ value includes a $2v(g_j)$ component either if it equals d_{j1} or d_{j2}. Thus, by summing up the $\hat{d_j}$ values on the left side of Inequality 8, a $2 \sum\limits_{g_j \in G} v(g_j)$ component will appear on the left side of Inequality 10. A $\hat{d_j}$ value has a further $-v(g_j)$ component exactly if it equals d_{j1}. A $\hat{d_j}$ value equals d_{j1} exactly if $g_j \in G'$. Thus $-v(g_j)$ has to be added to the left side of Inequality 10 for each $g_j \in G'$.

Since $|G| = |C_1|$, Inequality 10 can be reduced as follows:

$$2 \sum_{g_i \in G} v(g_i) - \sum_{g_j \in G'} v(g_j) \leq 2 \sum_{g_i \in G} v(g_i) - V \tag{11}$$

$$V \leq \sum_{g_j \in G'} v(g_j) \tag{12}$$

According to the assignments defined earlier in this proof, Inequality 9 can be transformed as follows:

$$\sum_{g_j \in G'} w(g_j) \leq W \tag{13}$$

The explanation for the left side of Inequality 13 is the following:

$\hat{c_j} = c_{j2} = 0$ exactly if $g_j \notin G'$ and $\hat{c_j} = c_{j1} = w(g_j)$ exactly if $g_j \in G'$. Thus, the left side of Inequality 13 will be the sum of those $w(g_j)$ values for which, $g_j \in G'$.

As a consequence of the transformations of Inequalities 8 and 9, our worst-case underperformance diagnostics problem will be solvable exactly if Inequalities 12 and

13 are true. However, as Inequality 12 is identical to Inequality 7 and Inequality 13 is identical to Inequality 6, our worst-case underperformance diagnostics problem is solvable if and only if the corresponding knapsack problem is solvable.

Since the transformation of the knapsack problem to an instance of the worst-case underperformance diagnostics problem can be carried out in $O(|G|)$ that is, in linear time and the knapsack problem is solvable if and only if the corresponding worst-case underperformance diagnostics problem is solvable, the knapsack problem is Karp reducible to the worst-case underperformance diagnostics problem.

And finally, since the knapsack problem is Karp reducible to the worst-case underperformance diagnostics problem and the worst-case underperformance diagnostics problem is in NP, the worst-case underperformance diagnostics problem is NP-complete.

□

4.3 ILP formulation of the Worst-Case Underperformance Diagnostics Problem

Since the worst-case underperformance diagnostics problem is NP-complete, the most effective known way to find its optimal solution is formulating it as an integer linear program, and solving it. The optimal solution in our case is the solution with the minimal cost. The integer linear program in our case will be the binary linear program (BLP) formulated in this subsection.

The binary program formulating the worst-case underperformance diagnostics problem is as follows, where $crr_i = \frac{i}{Gr}$, and $cst_i = Cost(\frac{i}{Gr})$:

Minimize:

$$\sum_{i:t_i \in T} \sum_{j=1}^{Gr} q_{ij} cst_j \tag{14}$$

Subject to:

$$\forall (i : t_i \in T) : \sum_{j=1}^{Gr} q_{ij} = 1 \tag{15}$$

$$\forall c_i \in C : \sum_{j:t_j \in c_i} d_j \sum_{k=1}^{Gr} q_{jk} crr_k \leq \frac{|c_i|}{CWR_{usr}} \tag{16}$$

$$\begin{gathered} \forall (i : t_i \in T) : \forall (j = 1, 2, \ldots, Gr) : \\ q_{ij} \in \{0, 1\} \end{gathered} \tag{17}$$

When solving the program, the values of the q_{ij} variables are being searched for. The value of each q_{ij} has to be set to 0 or 1 (Equation 17). Variables q_{ij}, where $j = 1, \ldots, Gr$ are used for choosing the value of correction factor q_i. As a solution of the BLP above, for each transition t_i, there is exactly one q_{ij} variable with the value of 1. All the other q_{ij} variables of transition t_i are set to 0. This is

a consequence of Equations 15 and 17. If the value of q_{ij} is 1 then q_i equals $\frac{j}{Gr}$, and thus, correcting the delay of t_i costs $Cost(\frac{j}{Gr})$.

As mentioned above, Equations 15 and 17 are responsible for choosing the value of q_i legally. According to these equations, each q_{ij} is 0 or 1 and for each t_i, exactly one q_{ij} equals 1, while the others equal 0. On the left side of Inequality 16, coefficient $\sum_{k=1}^{Gr} q_{jk}\, crr_k$ equals q_j the value of which is chosen from among the crr_k values by the appropriate q_{jk} variable set to 1. Thus, Inequality 16 means that the corrected delay of each cycle c_i has to be lower than or equal to $\frac{|c_i|}{CWR_{usr}}$ (this corresponds to Inequality 4). Finally, in the objective function (Formula 14), $\sum_{j=1}^{Gr} q_{ij}\, cst_j$ equals $Cost(q_i)$, which is the cost of correcting the delay of transition t_i. The value of $Cost(q_i)$ is chosen from among the cst_j values by the appropriate q_{ij} variable set to 1. Thus, the objective function expresses that the total cost of transition delay correction should be minimal.

Note: The problem can also be interpreted as a maximalization problem, where the maximal cost K is given and the task is to maximize CW_{usr}. Using the above notation, this problem can be formulated as follows:

Maximize:

$$CW_{usr} \tag{18}$$

Subject to:

$$\sum_{i:t_i \in T} \sum_{j=1}^{Gr} q_{ij}\, cst_j \leq K \tag{19}$$

$$\forall (i : t_i \in T) : \sum_{j=1}^{Gr} q_{ij} = 1 \tag{20}$$

$$\forall c_i \in C : \sum_{j:t_j \in c_i} d_j \sum_{k=1}^{Gr} q_{jk}\, crr_k \leq \frac{|c_i|}{CW_{usr}} \tag{21}$$

$$\begin{gathered} \forall (i : t_i \in T) : \forall (j = 1, 2, \ldots, Gr) : \\ q_{ij} \in \{0, 1\} \end{gathered} \tag{22}$$

The above formulation differs from the first formulation in two things. First, it sets an upper limit for the total cost and second, while it still requires each cycle delay to be lower than or equal to $\frac{|c_i|}{CW_{usr}}$, its objective is to maximize CW_{usr}. In the simulations presented in Section 5, we use the first formulation.

4.4 A Heuristic Solution for the Worst-Case Underperformance Diagnostics Problem

In this subsection, we introduce a heuristic algorithm for solving the worst-case underperformance diagnostics problem. The algorithm is optimized for the case when $Cost(x) = -\log_a x$ $(x \leq 1)$, an its objective is to minimize the cost by which CW_{usr} can be made greater than or equal to CWR_{usr}. Algorithm 1 shows how our heuristic method works. In the algorithm, r is the so called refreshing granularity (a large integer).

Algorithm 1: Heuristics for solving the worst-case underperformance diagnostics problem

input : T, C, Gr, CWR_{usr}, r
output: $\bigcup_{i:t_i \in T} q_i$

foreach $i : t_i \in T$ **do**
  $clist_i := \{j | t_i \in c_j\}$;
foreach $i : t_i \in T$ **do**
  $period_i := \lceil |clist_i| d_i r \rceil$;
foreach $i : t_i \in T$ **do**
  $q_i := \frac{1}{Gr}$;
if $\exists (c_i \in C) : \sum_{j:t_j \in c_i} \frac{d_j}{Gr} \leq \frac{|c_i|}{CWR_{usr}}$ **then**
  return "unsolvable";
$iteration := 1$;
while
$\exists (i : t_i \in T) : (q_i < 1 \wedge \forall (j \in clist_i) : (\sum_{k:t_k \in c_j \wedge k \neq i} q_k d_k) + (q_i + \frac{1}{Gr}) d_i \leq \frac{|c_j|}{CWR_{usr}})$
do
  foreach $i : t_i \in T$ **do**
    if $iteration \bmod period_i = 0 \wedge$
    $q_i < 1 \wedge \forall (j \in clist_i) : (\sum_{k:t_k \in c_j \wedge k \neq i} q_k d_k) + (q_i + \frac{1}{Gr}) d_i \leq \frac{|c_j|}{CWR_{usr}})$ **then**
      $q_i := q_i + \frac{1}{Gr}$;
  $iteration := iteration + 1$;
return $\bigcup_{i:t_i \in T} q_i$;

The algorithm first sets each correction factor q_i to its minimal value $\frac{1}{Gr}$ and then it increases each of these factors by $\frac{1}{Gr}$ more or less frequently, in a round robin manner, e.g. some of the correction factors might be increased in each round while some of the others might only be increased in every other round, etc. The key step of the algorithm is determining how frequently each q_i should be increased in order to keep the total cost minimal, while still keeping the delay of each cycle c_i lower than or equal to $\frac{|c_i|}{CWR_{usr}}$.

In the following, we are going to explain the algorithm in detail.

In lines 1 and 2, to each transition t_i, a set $clist_i$ is constructed, which stores references to each of the cycles that include t_i.

Lines 4 and 5 are the key steps of the algorithm. In these steps, for each transition t_i, we determine the value of $period_i$ which is the number of iterations that have to pass between two subsequent increasements of q_i. To explain why $\forall t_i : period_i := \lceil |clist_i| d_i r \rceil$, let us look at the following example.

Let us take a TCFMM consisting of two states s_0, and s_1, and two transitions t_0, and t_1, such that t_0 is originated in s_0 and destinated in s_1 while t_1 is originated in s_1 and destinated in s_0. Thus, in this TCFMM there is exactly one directed cycle. Let b denote the maximal allowed cycle delay we would like to achieve by correcting delays d_0, and d_1. In our case, $b = \frac{|c_1|}{CWR_{usr}} = \frac{2}{CWR_{usr}}$. Let us furthermore assume that in this example, the co-domains of q_0 and q_1 are continuous and the cost function is $Cost(x) = -\log_a x$.

Because of the continuous co-domains and the single directed cycle in the TCFMM, for the most cost-effective solution the corrected cycle delay will be equal to b and not lower than it. Thus, $d_0 q_0 + d_1 q_1 = b$. The latter equation means that $q_1 = \frac{b - d_0 q_0}{d_1}$.

The objective of the problem is to choose the appropriate q_i values that minimize $\sum_{i:t_i \in c_1} Cost(q_i)$. In our case this formula equals $\min_{q_0, q_1}(-(\log_a q_0 + \log_a q_1))$. The latter formula, using that $q_1 = \frac{b - d_0 q_0}{d_1}$, can be further transformed as follows:

$$\begin{gathered}
\min_{q_0,q_1}(-(\log_a q_0 + \log_a q_1)) \to \min_{q_0,q_1}(-\log_a q_0 q_1) \to \\
\min_{q_0}\left(-\log_a\left(-\tfrac{d_0}{d_1}q_0^2 + \tfrac{q_0 b}{d_1}\right)\right) \to \max_{q_0}\left(-\tfrac{d_0}{d_1}q_0^2 + \tfrac{q_0 b}{d_1}\right) \to \\
\min_{q_0}\left(\tfrac{d_0}{d_1}q_0^2 - \tfrac{q_0 b}{d_1}\right) \to \min_{q_0}\left(\sqrt{\tfrac{d_0}{d_1}}q_0 - \tfrac{b}{2d_1}\sqrt{\tfrac{d_1}{d_0}}\right)^2 - \tfrac{b^2}{4d_1^2}\tfrac{d_1}{d_0}
\end{gathered}$$

Since the second component of the above formula does not contain q_0, it is minimal if the first component $\left(\sqrt{\frac{d_0}{d_1}}q_0 - \frac{b}{2d_1}\sqrt{\frac{d_1}{d_0}}\right)^2$ is minimal. Since the first component is a square, it is minimal if it equals 0 that is, if $q_0 = \frac{b}{2d_1}\sqrt{\frac{d_1}{d_0}}\sqrt{\frac{d_1}{d_0}} = \frac{b}{2d_1}\frac{d_1}{d_0} = \frac{b}{2d_0}$. In this case $q_1 = \frac{b - d_0 q_0}{d_1} = \frac{b}{2d_1}$.

Thus, the cost of correcting the two transition delays will be minimal if $d_1 q_1 = \frac{b}{2}$, and $d_0 q_0 = \frac{b}{2}$, that is if the corrected delay values ($d_0 q_0$, and $d_1 q_1$) are equal. Based on this, we can suspect that in the case of a directed cycle which consists of more than two transitions, the cost of correcting the cycle delay is minimal if each corrected delay $d_i q_i$ value is equal. If however, the corrected transition delays of the cycle are equal, they equal $\frac{1}{CWR_{usr}}$. This is the consequence of Inequality 4, which takes the following shape for the optimal q_i values of this continuous problem (c_1 denotes the only transition cycle in the TCFMM):

$$\sum_{i:t_i \in c_1} d_i q_i = \frac{|c_1|}{CWR_{usr}} \tag{23}$$

As a consequence of the above, in the continuous case with a single directed cycle, $q_i = \frac{1}{d_0 CWR_{usr}}$.

Let us now return to the non-continuous case that our heuristic algorithm deals with. More precisely, let us see how to set the value of $period_i$ optimally. If the TCFMM consists of a single directed cycle, the appropriate correction factor for transition t_i can be achieved if the initial value $\frac{1}{Gr}$ of q_i is incremented in every $\lceil d_i r \rceil$-th round of the algorithm (see the explanation for r later). For example, if d_1 is twice as large as d_0 and thus, q_0 has to be around $2x$ and q_1 has to be around x then q_0 has to be incremented twice as frequently as q_1. To fulfill this requirement, one coefficient of $period_i$ is d_i which is responsible for making the corrected delay values approximately equal to each other.

Let us now assume that the TCFMM has multiple cycles and some of its transitions are included in more than one of these cycles (this is the general case). In this case, it is more cost-effective if we reduce the delays of transitions included in many cycles by a bigger amount than the delays of transitions included in fewer cycles. The reason for this is that if we reduce the delay of a transition, which is included in n cycles, then n cycle delays will be reduced. This way the left side of n instances of Inequality 4 will be reduced, while the cost of this reduction will not be multiplied by n. Based on this, we can suspect that by adding $|clist_i|$ as a coefficient to $period_i$ for each transition t_i, the total cost of delay reduction will be closer to optimal. We have confirmed this suspicion by running simulations.

The reason for including refreshing granularity r in $period_i$ is that in order to make $period_i$ an integer value, the ceiling value of $|clist_i|d_i$ is taken. We have chosen the ceiling value instead of the floor value to avoid the illegal case when $period_i = 0$. If $|clist_i|d_i$ is a small integer, then by taking its ceiling value, some of the accuracy of $|clist_i|d_i$ is lost. If however, we multiply $|clist_i|d_i$ by r and then take the ceiling value of $|clist_i|d_i r$, then the bigger r is, the more of this otherwise lost accuracy can be preserved. The value to be chosen for r is however, not independent from $|clist_i|d_i$. As the smaller $|clist_i|d_i$ is, the greater r has to be to preserve the same amount of accuracy. During our simulations presented in Section 5, we required r to be larger than or equal to $\lceil \frac{100}{\min\limits_{i:t_i \in T} |clist_i|d_i} \rceil$.

After setting the value of $period_i$, in lines 6 and 7, the algorithm initializes each q_i to its smallest possible value $\frac{1}{Gr}$. Then in lines 8 and 9, the algorithm checks whether using these lowest possible values of the correction factors, Inequality 4 is fulfilled for each cycle or not. If not, the problem is unsolvable, since each q_i has taken its smallest possible value and thus, no transition delay can be further reduced.

From line 10, the algorithm runs iterations. While there exists a transition t_i the correction factor q_i of which can be augmented by $\frac{1}{Gr}$ without violating any instances of Inequality 4 and without q_i getting bigger than 1 (which is the greatest possible value of each q_i), the algorithm starts a new iteration. In each iteration, the algorithm selects each transition t_i for which the iteration count is divisible by $period_i$. If for a selected transition, q_i is yet lower than 1, and q_i can be increased by $\frac{1}{Gr}$ without violating any instances of Inequality 4, q_i is increased by $\frac{1}{Gr}$. The

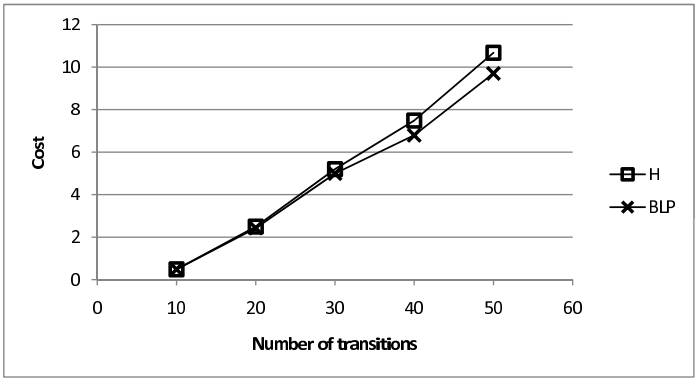

Figure 3: Costs of solutions, $Gr = 2$, $CWR_{usr} = 1.0$

iterations go on until no further correction factor can be increased.

5 Simulation Results

In this subsection, we present simulation results comparing the efficiency of our heuristic algorithm (denoted by H in the figures) to that of solving the fisrt BLP (denoted by BLP in the figures) formulated in Subsection 4.2, which always finds the optimal solution that is, the lowest cost solution.

The simulations were run on multiple TCFMMs, each one having 10 states. The structure of the TCFMMs (i.e. their states and state transitions without the delays assigned to each transition) were built incrementally. The first TCFMM structure has 10 transitions, while each of the others were constructed by taking the previous TCFMM structure and adding 10 random transitions to it. The largest structure has 50 transitions. From each TCFMM structure, we have generated 10 different TCFMMs by assigning random transition delay values to the structures. In the following, a group of TCFMMs means the TCFMMs having the same number of transitions.

During the simulations, we have measured the average time and cost needed to correct the transition delays of the TCFMMs using each method, as a function of the number of transitions in the TCFMM. The average cost and running time values were calculated for each group of TCFMMs. Since the set of cycles is an input parameter for both the BLP and the heuristic algorithm, all cycles in the TCFMMs had to be found before executing any of the methods. Thus, each time value plotted in the following figures is the average amount of time needed to run the methods plus the amount of time needed to find the cycles in the corresponding group of TCFMMs. For finding the cycles, we used an iterative deepening depth-first search. Transition delays were generated with uniform distribution on interval $[0.5, 1.5]$ that is, the mean transition delay is $d_{mean} = 1.0$.

During the simulations, $Cost(x) = -\ln x$.

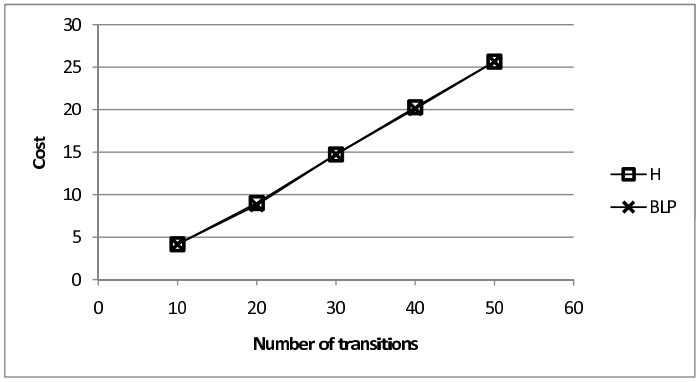

Figure 4: Costs of solutions, $Gr = 2$, $CWR_{usr} = 1.5$

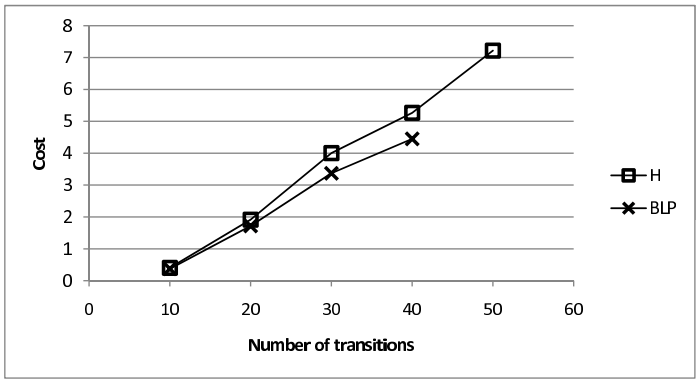

Figure 5: Costs of solutions, $Gr = 4$, $CWR_{usr} = 1.0$

Figure 3 shows the average costs of correcting the transition delays using each method, where $Gr = 2$, and $CWR_{usr} = 1.0$. Figure 4 plots the average costs of each method, where $Gr = 2$, and $CWR_{usr} = 1.5$. According to Figure 3, where $CWR_{usr} = \frac{1}{d_{mean}}$, there is no significant difference between the cost-efficiency of the two methods. The cost of the heuristic solution is by 5,4 percent higher than the optimal cost, in average. In the case of Figure 4 where $CWR_{usr} = \frac{1.5}{d_{mean}}$ however, the cost of the heuristic algorithm is only by 0,7 percent higher than the optimum, in average.

Figure 5 shows the average costs of the two different methods, where $Gr = 4$, and $CWR_{usr} = 1.0$. Figure 6 shows the average costs of the solutions found by each method, where $Gr = 4$, and $CWR_{usr} = 1.5$. According to Figure 5, the cost obtained by the heuristics is by 14 percent higher than the optimum, in average. However, as can be seen in Figure 6, if $CWR_{usr} = \frac{1.5}{d_{mean}}$ then the cost of the heuristics is higher than the optimal cost by only 5 percent.

Figure 7 shows the costs obtained by the two methods in the case where $Gr = 10$,

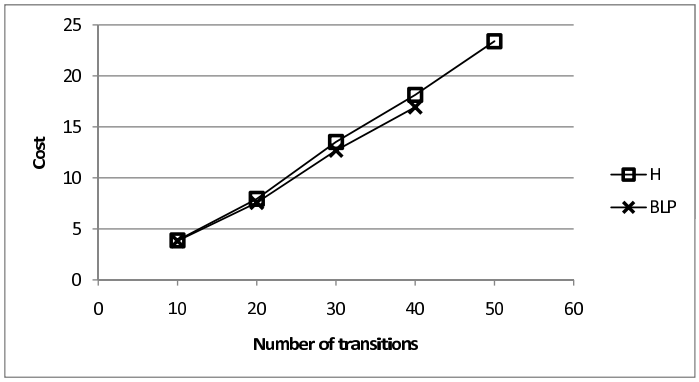

Figure 6: Costs of solutions, $Gr = 4$, $CWR_{usr} = 1.5$

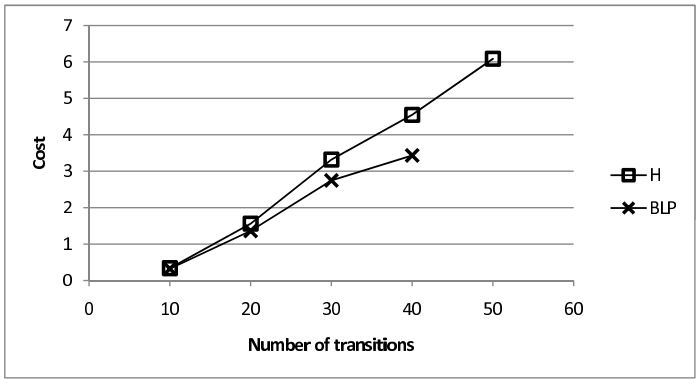

Figure 7: Costs of solutions, $Gr = 10$, $CWR_{usr} = 1.0$

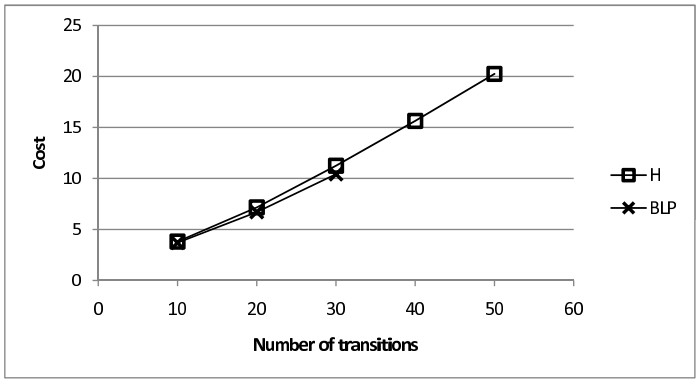

Figure 8: Costs of solutions, $Gr = 10$, $CWR_{usr} = 1.5$

Table 1: Rates of costs, $Cost(x) = -\ln x$

Gr	CWR_{usr}	$\frac{C(heuristics)}{C(BLP)}$
2	1.0	1.0545
2	1.5	1.007
4	1.0	1.1408
4	1.5	1.0497
10	1.0	1.1815
10	1.5	1.0598

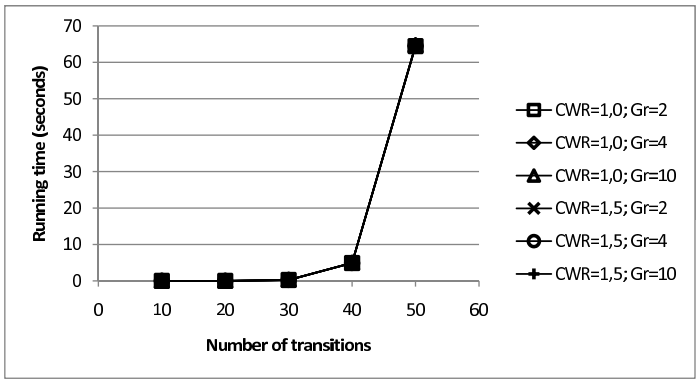

Figure 9: Running times of the heuristic algorithm

and $CWR_{usr} = 1.0 = \frac{1}{d_{mean}}$, while Figure 8 shows tis comparison for the case where $Gr = 10$, and $CWR_{usr} = 1.5$. As can be seen in Figure 7, if $CWR_{usr} = 1.0$, the cost of the solution obtained by the heuristic method is higher by 18 percent than the optimal cost, in average. According to Figure 8, in average, the cost of the heuristics is by only 6 percent higher than the optimal cost.

Table 1 shows the average rates of costs of the two methods. In the table, C denotes cost.

Figure 9, and Table 2 show the average running time of the heuristic solution (plus the amount of time needed for finding the transition cycles in the TCFMMs)

Table 2: Running times of the heuristic algorithm in seconds

CWR_{usr}	Gr	$\|T\| = 10$	$\|T\| = 20$	$\|T\| = 30$	$\|T\| = 40$	$\|T\| = 50$
$1,0$	2	$0,006$	$0,0108$	$0,2592$	$4,9084$	$64,4324$
$1,0$	4	$0,006$	$0,0112$	$0,2592$	$4,9204$	$64,5144$
$1,0$	10	$0,0064$	$0,012$	$0,264$	$4,9516$	$64,712$
$1,5$	2	$0,0068$	$0,0113$	$0,259$	$4,909$	$64,432$
$1,5$	4	$0,0064$	$0,0112$	$0,258$	$4,914$	$64,4692$
$1,5$	10	$0,0064$	$0,012$	$0,26$	$4,9376$	$64,6112$

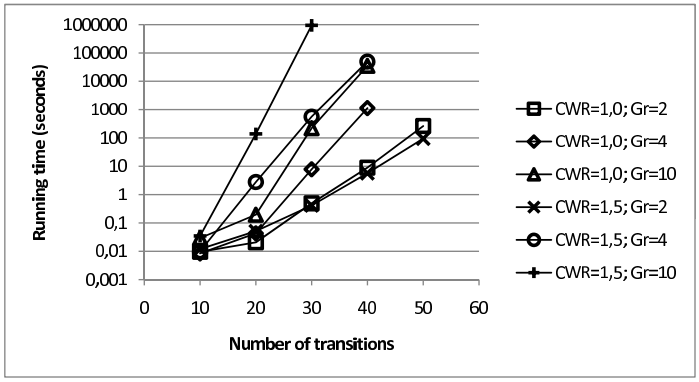

Figure 10: Running times of the BLP

Table 3: Running times of the BLP in seconds

CWR_{usr}	Gr	$\|T\| = 10$	$\|T\| = 20$	$\|T\| = 30$	$\|T\| = 40$	$\|T\| = 50$
1,0	2	0,0096	0,0208	0,4932	8,9786	263,681
1,0	4	0,0084	0,0428	7,85	1132,38	
1,0	10	0,0292	0,196	221,1	35018,4	
1,5	2	0,012	0,0513	0,415	5,695	94,842
1,5	4	0,01	2,7813	564,205	49765,9	
1,5	10	0,0348	138,793	942004,119		

as a function of the number of transitions in the TCFMMs, for different simulation scenarios. In the table, $|T|$ denotes the number of transitions in the TCFMM. As it can be seen in the figure and the table, the running time of the heuristic solution does not differ significantly for the different simulation scenarios (the markers of each curve are almost exactly on top of each other). The reason for this is that the running time of the heuristic algorithm itself is overweighed by the amount of time needed to find the transition cycles in the TCFMMs. As it can also be seen, the running time of the heuristics is reasonable even for extremely dense TCFMMs.

Figure 10 having a logarithmic vertical axis, and Table 3 show the average amount of time needed for solving the BLP (plus the amount of time needed for finding the transition cycles in the TCFMMs) as a function of the number of transitions in the TCFMMs. As the figure and the table show, the running time of the BLP increases significantly as Gr or CWR_{usr} grows. Thus, solving the BLP is only a reasonable choice for lower Gr and CWR_{usr} values. However, in the cases where the running time of the BLP is the highest, the heuristic algorithm is capable of calculating a solution the cost of which is not significantly higher than that of the BLP.

6 Summary

In this paper, we have proposed performance diagnostic methods. These methods attempt to determine how to increase the performance of the SUT if according to its performance test, it is unable to serve the required worst-case number of messages per second.

The increasement is achieved by decreasing transition delays and thus, increasing the number of messages per second that the SUT is able to process in worst case. Each transition delay can be decreased by discrete amounts and each delay reduction has a cost, which should be as low as possible. The reduced delay of transition t_i will be $d_i q_i$, where q_i is the so-called correction factor of transition t_i.

By reducing an arbitrary instance of the NP-complete knapsack problem, we have proven that this, so-called worst-case underperformance diagnosctics problem is NP-complete and formulated it as a binary linear program. We have also given a heuristic approach which works by first choosing the lowest possible (and most expensive) value of each correction factor and then by incrementing the correction factors more or less frequently. By incrementing correction factors, the cost of performance correction is decreased. The number of iterations that has to elapse between two subsequent increasements of a given transition is determined by a weight assigned to the transition.

We have compared the efficiency of solving the binary linear program to those of our heuristics and found that the latter performs efficiently in those cases where the former has an unreasonable running time.

7 Future Work

In our future work, we are going to extend the worst-case underperformance diagnostics problem for a more general case, in which the legal correction factors q_i vary for different transitions (e.g. the legal correction factor values for transition t_1 are 0.6 and 0.8, while the legal correction factor values for transition t_2 are 0.1, 0.15 and 0.3, etc).

We are also going to deal with another extension of the worst-case underperformance diagnostics problem in which, instead of decreasing transition delays, the performance of *resources* of the system can be increased by different amounts. As a result of increasing the performance of a resource, the delays of a set of transitions decrease by different amounts. Each resource performance increasement has a cost, and the task is to determine how to increase the performance of each resource in order to make the system meet Inequality 2 at minimal cost.

Acknowledgments

This paper has been (partially) supported by HSNLab, Budapest University of Technology and Economics, http://www.hsnlab.hu.

References

[1] Lee, D. and Yannakakis, M. Principles and Methods of Testing Finite State Machines – A Survey In *Proceedings of the IEEE vol. 84 issue 8*, pages 1090–1123, 1996.

[2] Cavalli, A. R., Dorofeeva, R., El-Fakih, K., Maag, S. and Yevtushenko, N. FSM-based conformance testing methods: A survey annotated with experimental evaluation In *Information and Software Technology vol. 52 issue 12.*, pages 1286–1297, 2010.

[3] Feng, C., Lombardi, F., Shen, Y. and Sun, X. Advanced Series in Electrical and Computer Engineering - Vol. 12, Protocol Conformance Testing Using Unique Input/Output Sequences World Scientific Publishing, River Edge, NJ, 1997.

[4] ISO/IEC 9646-1: Information Technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 1: General concepts, 1994.

[5] Chul, K. and Song, J. S. Test Sequence Generation Methods for Protocol Conformance Testing In *Proc. of the Eighteenth Annual International Computer Software and Applications Conference*, pages 169–174, 1994.

[6] ITU-T ITU-T Recommendation Z.500 – Framework on formal methods in conformance testing, 1997.

[7] Boroday, S., Groz, R. and Petrenko, A. Confirming configurations in EFSM In *Proc. of the IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XII) and Protocol Specification, Testing and Verification (PSTV XIX)*, pages 5–24, 1999.

[8] Lee, D. and Yannakakis, M. Testing Finite-State Machines: State Identification and Verification In *IEEE Transactions on Computing vol. 43 issue 3.*, pages 306–320, 1994.

[9] Aho, A.V. and Lee, D. Efficient algorithms for constructing testing sets, covering paths, and minimum flows In *AT&T Bell Laboratories Technical Memorandum*, 1987.

[10] Tretmans, G. J. Test Generation with Inputs, Outputs, and Quiescence In *Proc. Tools and Algorithms for Construction and Analysis of Systems, Second International Workshop, TACAS*, pages 127–146, 1996.

[11] Brinksma, E., Tretmans, J. and Verhaard, L. A Framework for Test Selection In *Proc. IFIP WG6.1 11th Int. Symp. on Protocol Specification, Testing, and Verification*, pages 233–248, 1991.

[12] Amalou, M., Fujiwara, S., Ghedamsi, A. and Khendek, F. Test Selection Based on Finite State Models In *IEEE Transactions on Software Engineering vol. 17 issue 6.*, pages 591–603, 2002.

[13] Dahbura, A.T., Sabnani, K.K. and Uyar, M.U. Formal Methods for Generating Protocol Conformance Test Sequences In *Proceedings of the IEEE vol. 78 issue 8*, pages 1317–1326, 2002.

[14] Csopaki, G., Kovacs, G., Pap, Z. and Tarnay, K. Iterative Automatic Test Generation Method for Telecommunication Protocols In *Computer Standards & Interfaces vol. 28 issue 4.*, pages 412–427, 2006.

[15] Bochmann, G.V., Dssouli, R. and Ghedamsi, A. Multiple Fault Diagnosis for Finite State Machines In *INFOCOM'93. Proc. Twelfth Annual Joint Conference of the IEEE Computer and Communications Societies. Networking: Foundation for the Future. IEEE*, pages 782–791, 2002.

[16] Bause, F., Kabutz, H., Kemper, P. and Kritzinger, P. SDL and Petri Net Performance Analysis of Communicating Systems In *Proc. of the 15th International Symposium on Protocol Specification, Testing and Verification*, pages 269–282, 1995.

[17] ITU-T Recommendation Z.100 – Specification and Description Language (SDL), 1994.

[18] El-Kilani, W. S., El-Wahed, W. F. A. and Youness, O. S. A Behavior and Delay Equivalent Petri Net Model for Performance Evaluation of Communication Protocols In *Computer Communications, vol. 31 issue 10.*, pages 2210–2230, 2008.

[19] Marsan, M. A. Stochastic Petri Nets: An Elementary Introduction In *Advances in Petri Nets, Lecture Notes in Computer Science, vol. 424*, pages 1–29, 1990.

[20] Murata, T. Petri Nets: Properties, Analysis and Applications In *Proceedings of the IEEE vol. 77. issue 4.*, pages 541–580, 1989.

[21] Al-Obaidan, A., El-Karaksy, M. R. and Nouh, A. S. Performance Analysis of Timed Petri Net Models for Communication Protocols: A Methodology and Package In *Computer Communications vol. 13 issue 2.*, pages 73–82, 1990.

[22] Chiola, G., Fumagalli, A. and Marsan, M. A. Timed Petri Net Model for the Accurate Performance Analysis of CSMA/CD Bus LANs In *Computer Communications vol 10. issue 6.*, pages 304–312, 1987.

[23] Schieferdecker, I., Stepien, B. and Rennoch, A. PerfTTCN, a TTCN Language Extension for Performance Testing In *Proc. of the IFIP TC6 10th International Workshop on Testing of Communicating Systems*, pages 21–36, 1997.

[24] ISO/IEC 9646-1: Information Technology - Open Systems Interconnection - Conformance Testing Methodology and Framework - Part 3: The Tree and Tabular Combined Notation, 1995.

[25] ETSI ES 201 873-1 ver. 4.2.1: Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language 2010.

[26] Dai, Z. R., Grabowski, J. and Neukirchen, H. TimedTTCN-3 - A Real-Time Extension for TTCN-3 In *Testing of Communicating Systems*, pages 407–424, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.

[27] Mingwei, X. and Jianping, W. A formal approach to protocol performance testing In *Journal of Computer Science and Technology vol. 14 issue 1*, pages 81–87, 1999.

[28] ISO/IEC 9646-3 AM. 1: Information technology - OSI conformance testing methodology and framework - Concurrent TTCN, 1993.

[29] Csondes, T. and Eros, L. An Automatic Performance Testing Method Based on a Formal Model for Communicating Systems In *Proc. of the 18th International Workshop on Quality of Service (IWQoS)*, 2010.

[30] Garey, M. R. and Johnson, D. S. Computers and Intractability; A Guide to the Theory of NP-Completeness W. H. Freeman & Co., San Francisco, CA, 1990.

Received 23rd May 2011