Volume 20 Number 4

ACTA CYBERNETICA

Szeged, 2012

ACTA CYBERNETICA

Information for authors. Acta Cybernetica publishes only original papers in the field of Computer Science. Manuscripts must be written in good English. Contributions are accepted for review with the understanding that the same work has not been published elsewhere. Papers previously published in conference proceedings, digests, preprints are eligible for consideration provided that the author informs the Editor at the time of submission and that the papers have undergone substantial revision. If authors have used their own previously published material as a basis for a new submission, they are required to cite the previous work(s) and very clearly indicate how the new submission offers substantively novel or different contributions beyond those of the previously published work(s). Each submission is peer-reviewed by at least two referees. The length of the review process depends on many factors such as the availability of an Editor and the time it takes to locate qualified reviewers. Usually, a review process takes 6 months to be completed. There are no page charges. An electronic version of the puplished paper is provided for the authors in PDF format.

Manuscript Formatting Requirements. All submissions must include a title page with the following elements:

- title of the paper
- author name(s) and affiliation
- name, address and email of the corresponding author
- An abstract clearly stating the nature and significance of the paper. Abstracts must not include mathematical expressions or bibliographic references.

References should appear in a separate bibliography at the end of the paper, with items in alphabetical order referred to by numerals in square brackets. Please prepare your submission as one single PostScript or PDF file including all elements of the manuscript (title page, main text, illustrations, bibliography, etc.). Manuscripts must be submitted by email as a single attachment to either the most competent Editor, the Managing Editor, or the Editor-in-Chief. In addition, your email has to contain the information appearing on the title page as plain ASCII text. When your paper is accepted for publication, you will be asked to send the complete electronic version of your manuscript to the Managing Editor. For technical reasons we can only accept files in LaTeX format.

Subscription Information. Acta Cybernetica is published by the Institute of Informatics, University of Szeged, Hungary. Each volume consists of four issues, two issues are published in a calendar year. Subscription rates for one issue are as follows: 5000 Ft within Hungary, €40 outside Hungary. Special rates for distributors and bulk orders are available upon request from the publisher. Printed issues are delivered by surface mail in Europe, and by air mail to overseas countries. Claims for missing issues are accepted within six months from the publication date. Please address all requests to:

Acta Cybernetica, Institute of Informatics, University of Szeged
P.O. Box 652, H-6701 Szeged, Hungary
Tel: +36 62 546 396, Fax: +36 62 546 397, Email: acta@inf.u-szeged.hu

Web access. The above informations along with the contents of past issues are available at the Acta Cybernetica homepage http://www.inf.u-szeged.hu/actacybernetica/ .

Ferenc Gécseg
Department of Computer Algorithms
and Artificial Intelligence
University of Szeged
Szeged, Hungary
gecseg@inf.u-szeged.hu

Jozef Gruska
Institute of Informatics/Mathematics
Slovak Academy of Science
Bratislava, Slovakia
gruska@savba.sk

Tibor Gyimóthy
Department of Software Engineering
University of Szeged
Szeged, Hungary
gyimothy@inf.u-szeged.hu

Helmut Jürgensen
Department of Computer Science
Middlesex College
The University of Western Ontario
London, Canada
helmut@csd.uwo.ca

Zoltan Kato
Department of Image Processing
and Computer Graphics
Szeged, Hungary
kato@inf.u-szeged.hu

Alice Kelemenová
Institute of Computer Science
Silesian University at Opava
Opava, Czech Republic
Alica.Kelemenova@fpf.slu.cz

László Lovász
Department of Computer Science
Eötvös Loránd University
Budapest, Hungary
lovasz@cs.elte.hu

Gheorghe Păun
Institute of Mathematics of the
Romanian Academy
Bucharest, Romania
George.Paun@imar.ro

András Prékopa
Department of Operations Research
Eötvös Loránd University
Budapest, Hungary
prekopa@cs.elte.hu

Arto Salomaa
Department of Mathematics
University of Turku
Turku, Finland
asalomaa@utu.fi

László Varga
Department of Software Technology
and Methodology
Eötvös Loránd University
Budapest, Hungary
varga@ludens.elte.hu

Heiko Vogler
Department of Computer Science
Dresden University of Technology
Dresden, Germany
Heiko.Vogler@tu-dresden.de

Gerhard J. Woeginger
Department of Mathematics and
Computer Science
Eindhoven University of Technology
Eindhoven, The Netherlands
gwoegi@win.tue.nl

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

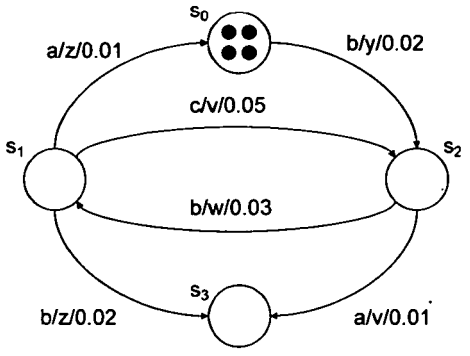

Figure 1: The TCFMM model

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

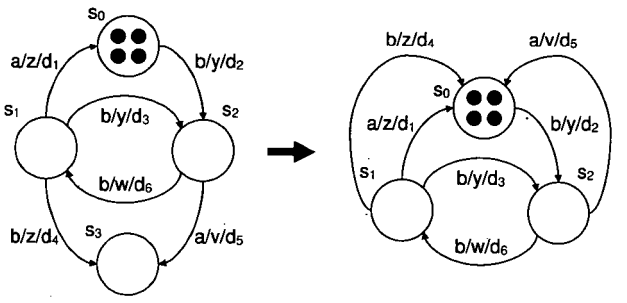

Figure 2: Redirecting transitions from terminating states

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:

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

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 $Cost : (0, 1] \rightarrow \mathbb{R}^+$, $Cost(1) = 0$ 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:

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

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:

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

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 $2 \sum_{g_j \in G} v(g_j)$ 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:

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

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

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

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:

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

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

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

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

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: $\bigcup_{i:t_i \in T} q_i$

1 **foreach** $i : t_i \in T$ **do**
2 | $clist_i := \{j | t_i \in c_j\}$;
3 **foreach** $i : t_i \in T$ **do**
4 | $period_i := \lceil |clist_i| d_i r \rceil$;
5 **foreach** $i : t_i \in T$ **do**
6 | $q_i := \frac{1}{Gr}$;
7 **if** $\exists (c_i \in C) : \sum_{j:t_j \in c_i} \frac{d_j}{Gr} \leq \frac{|c_i|}{CWR_{usr}}$ **then**
8 | **return** "unsolvable";
9 $iteration := 1$;
10 **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
11 | **foreach** $i : t_i \in T$ **do**
12 | | **if** $iteration$ mod $period_i = 0 \wedge$
$q_i < 1 \wedge \forall (j \in clist_i) : (\sum_{k:t_k \in c_j \wedge k \neq i} q_k d_k) + (q_i + \frac{1}{Gr}) d_i \leq \frac{|c_j|}{CWR_{usr}})$ **then**
13 | | | $q_i := q_i + \frac{1}{Gr}$;
14 | $iteration := iteration + 1$;
15 **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:

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

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

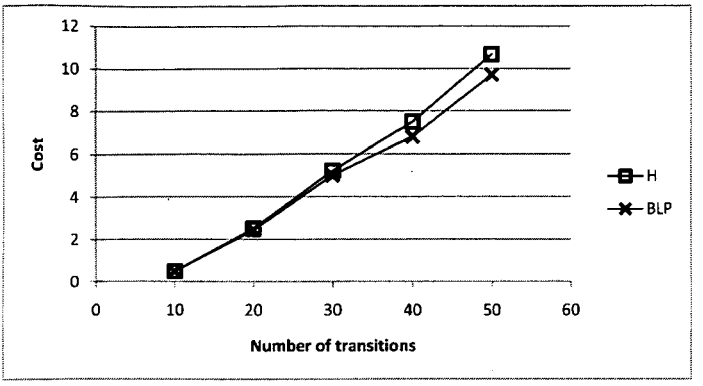

Figure 3: Costs of solutions, $Gr = 2$, $CWR_{usr} = 1.0$

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, $Cost(x) = -\ln x$.

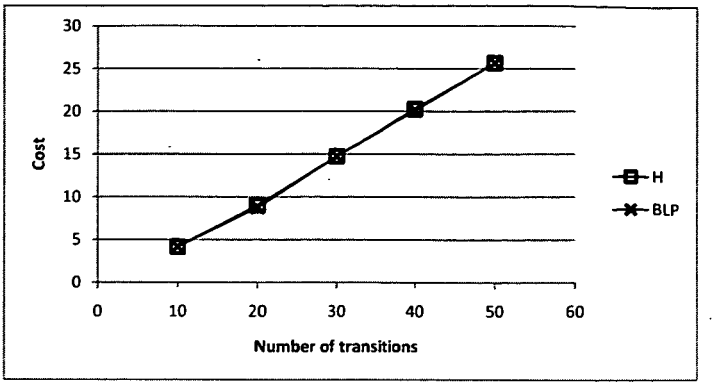

Figure 4: Costs of solutions, $Gr = 2$, $CWR_{usr} = 1.5$

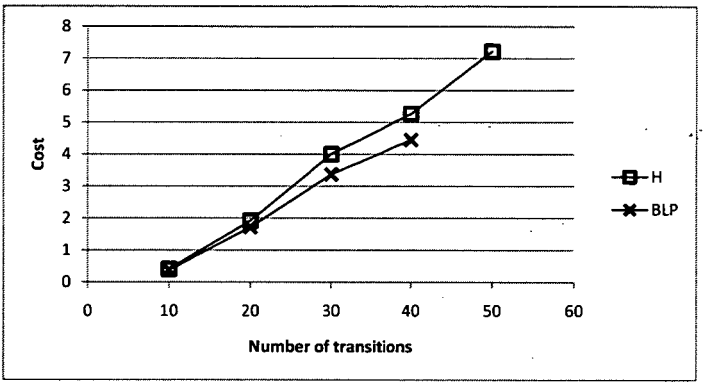

Figure 5: Costs of solutions, $Gr = 4$, $CWR_{usr} = 1.0$

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 $Gr = 10$,

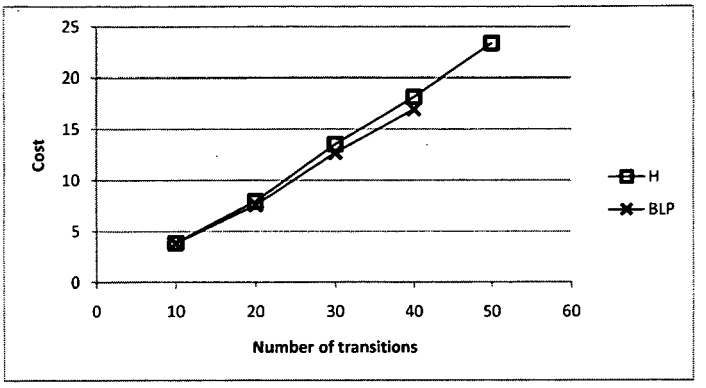

Figure 6: Costs of solutions, $Gr = 4$, $CWR_{usr} = 1.5$

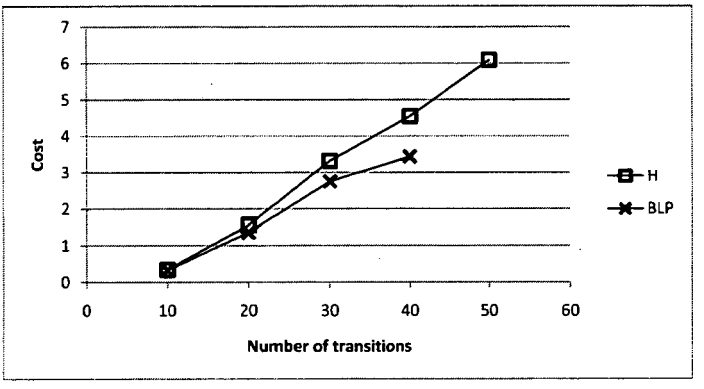

Figure 7: Costs of solutions, $Gr = 10$, $CWR_{usr} = 1.0$

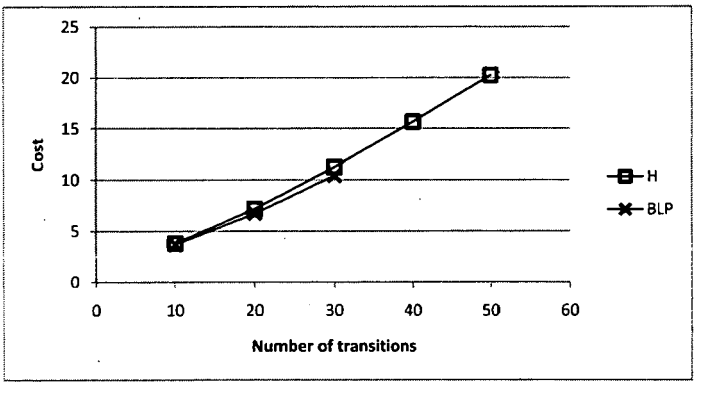

Figure 8: Costs of solutions, $Gr = 10$, $CWR_{usr} = 1.5$

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

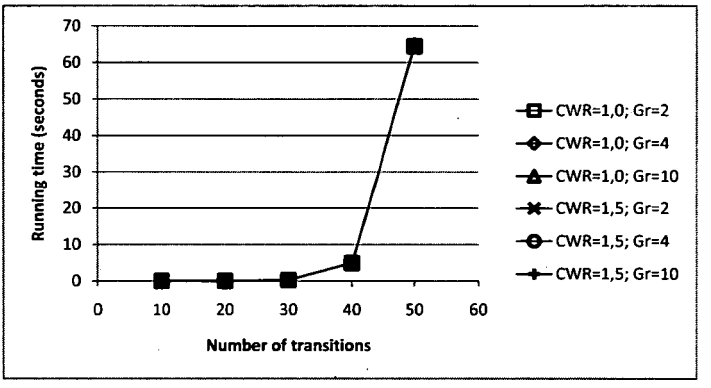

Figure 9: Running times of the heuristic algorithm

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

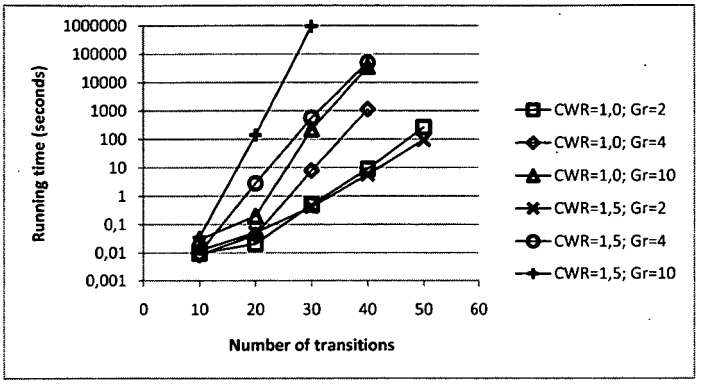

Figure 10: Running times of the BLP

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

Acta Cybernetica **20** (2012) 483–497.

Understanding Program Slices

Ákos Hajnal* and István Forgács†

Abstract

Program slicing is a useful analysis for aiding different software engineering activities. In the past decades, various notions of program slices have been evolved as well as a number of methods to compute them. By now program slicing has numerous applications in software maintenance, program comprehension, reverse engineering, program integration, and software testing. Usability of program slicing for real world programs depends on many factors such as precision, speed, and scalability, which have already been addressed in the literature. However, only a little attention has been brought to the practical demand: when the slices are large or difficult to understand, which often occur in the case of larger programs, how to give an explanation for the user why a particular element has been included in the resulting slice. This paper describes a reasoning method about elements of static program slices.

Keywords: data flow analysis, static program slicing, reasoning

1 Introduction

Program slicing is a source code analysis technique proposed by Mark Weiser [35] capable of automatically identifying the set of program statements, called the *slice*, which may affect the values of the selected variables at a program point of interest, called the *slicing criterion*. Program slicing uses dependence analysis that examines the source code to trace control- and data flow to determine the statements that belong to the slice.

Weiser's original method – motivated to aid debugging activities – has been classified later as a "backward static" program slicing technique. *Backward*, because in constructing the slice, statements affecting the selected statement are traced backwards, and *static*, because the analysis is made without having specified any particular program execution, i.e. all possible program executions are taken into account. Forward static program slicing determines the part of the program that is directly or indirectly affected by the selected statement.

*Computer and Automation Research Institute Hungarian Academy of Sciences, E-mail: ahajnal@sztaki.hu

†4D Soft Ltd., E-mail: forgacs@4dsoft.hu

Since Weiser's method other variants of program slicing have been evolved such as dynamic slicing [30, 2], quasi-static slicing [34], conditioned slicing [8], amorphous slicing [22], hybrid slicing [19], and relevant slicing [20]. In the past decades, numerous applications of program slicing have been proposed in different areas of software engineering, including software maintenance [16, 15, 10, 11], program comprehension [12, 24], reverse engineering [9], program integration [27, 7], and software testing [18, 23, 4, 5, 14, 25, 26].

Program slicing allows the users to focus on the selected aspects of semantics by breaking the whole program into smaller pieces, and when these slices are small they can be more easily maintained. However, lager program slices, but even slices containing only some tens of program instructions can be very difficult to understand. As William Griswold [17] pointed out in his talk: *Making Slicing Practical: The Final Mile*, one of the problems why slicers are not widely used is that it is not enough to dump the results onto the screen without explanation.

Slices computed based on execution traces (dynamic) are typically smaller than the ones that consider all possible program executions (static). Furthermore, as a particular execution history is available during dynamic slicing, the chain of dependences caused a given program statement to be included in the slice can be more easily discovered. This is not the case in static slicing, where neither a particular dependence chain nor an execution trace covering these dependences are presented. Some applications such as program comprehension, re- and reverse engineering rely on static slicing, and it may occur that code under analysis cannot be even compiled and run (legacy systems, program under development).

Static program slicing gives a wider view to the connected parts of the program code, which is essential in program comprehension or at extracting reusable functions from legacy systems – considering all possible program executions. Note that without an automated slicing tool revealing dependences in the program text is very labor-intensive, tedious, and time consuming task. These techniques calculate the set of statements that directly or indirectly affect (or affected by) the slicing criterion. However, beyond claiming that there is dependence between the slicing criterion and the computed slice element, no explanation of the result is provided, which could help in understanding the effects between different parts of the program code by the human users.

For example, in regression testing, one can use static program slicing to determine those parts of the code that are affected by the program modification. It can occur that one or more slice elements fall out of the software component that the change supposed to be influenced, so the user may be curious how the effect has reached that point. By showing a particular chain of dependences from the slicing criterion to the selected slice element the user could be convinced that the influence indeed exists, and either there is an unforeseen, undesired side effect of the modification, or this effect has not been taken into consideration at determining the impact of the change.

The more precise the applied slicing technique the less the resulting slice sizes are. There are no fully precise static slicing methods for real programming languages, so *false positives*, i.e. slice elements identified on dependences that actually

cannot occur during real program executions are unavoidable. One source of such imprecision is due to following non-realizable program paths during the analysis (paths along which procedure calls and returns are incorrectly nested). By applying *context-sensitive* techniques, these false positives can be filtered out. Other sources of imprecision are due to infeasible program paths (no such program input that results in the execution of the traversed conditional branches) and programming language constructs that make impossible to recover the precise flow of data statically (use of pointers, dynamic constructs). The latter two problems are not solvable in general; static slicing techniques typically use a conservative approach to provide safe results (consider all potential but not necessarily "real" effects).

In this case, reasoning about slice elements could help programmers to recognize false positives. In regression testing, for example, an unexpected impact of a program change may be proven to be false, when the presented chain of dependences is infeasible (it can be realized along infeasible paths only), and it is rejected by a human user. This is a manual process, but it can still be less expensive than retesting all the slicer indicated parts of the code.

This paper concerns with the token propagation-based context-sensitive, static program slicing technique [21], and proposes a method to reason about the computed slice elements. Reasoning means showing a specific *dependence chain* – along with control-flow information – from the slicing criterion to the selected slice element.

The rest of the paper is organized as follows. Section 2 provides an overview of the necessary concepts, and summarizes the basic rules of the token propagation-based slicing method. Section 3 describes how a dependence trace for the slice elements can be derived by computing a particular dependence chain. Section 4 discusses the related work. Finally, Section 5 concludes the paper.

2 Background

Computer programs can be represented by directed graphs called *control flow graphs* (CFGs). Control flow graphs are constructed by assigning a directed graph to each procedure (*intra*procedural control flow graph) with unique entry and exit nodes, in which nodes correspond to the statements and predicates in the procedure, and edges represent the possible flow of control. A call statement is represented by two nodes, a *call site* and a *return site*, which are linked to the entry and exit node of the called procedure, respectively (*inter*procedural control flow graph). We refer to the related call site c and return site r using the *callSiteOf*, *returnSiteOf* operators, such that, c=*callSiteOf*(r) and r=*returnSiteOf*(c).

Variable references and assignments are referred to as *uses* and *definitions* in nodes. A definition is *influenced* by a use in the same node if the assigned value is dependent on the value of the referenced variable. A path containing no definition for a variable v (excluding start and end nodes) is a *definition-clear path* with respect to variable v. The definition of v in node n and the use of the same variable in node m form a *definition-use pair* if there is a definition-clear path with respect

to v from n to m. Node m is said to be (directly) *data dependent* on node n.

A statement S is *control dependent* on predicate P if the outcome of P determines whether S executes. Intuitively it means that statements contained by conditionally executed branches are control dependent on the predicate. Control dependent procedure calls extend the control scope to statements in the called procedure(s). There are different definitions and computations of control dependence, the particular notion is however orthogonal to how to compute the slice. We assume that (intraprocedural) control dependences are available in the program graph, represented by *control edges*. We treat interprocedural control dependences indirectly, by introducing control edges from call sites to procedure entry nodes, and from entry to other nodes in the procedure – except entry- and exit nodes, as shown in Figure 1.

The transitive flow of data and control dependences form a *dependence chain*, which is a sequence of nodes n_1, n_2, ..., n_k, where node n_{i+1} is either directly data- or control dependent on node n_i for all i, $1 \leq i \leq k-1$. Nodes n_2, n_3, ..., n_k are said to be *affected* by node n_1. A path p *covers* the dependence chain if it goes through chain nodes n_1, n_2, ..., n_k, and each subpath of p between nodes n_i and n_{i+1} is either definition-clear with respect to the variable defined at n_i (data dependence), or all the nodes of the subpath are control dependent on n_i (control dependence), respectively. The dependence chain is realizable if it can be covered by a realizable path.

A slicing criterion is a pair $C=<I, V>$, where I is a program point and V is a subset of program variables. The backward static program slice S with respect to slicing criterion C consists of all the parts of the program that have direct or indirect effect on the values computed for variables V at I. In forward static program slicing, statements depending on the slicing criterion C are computed, where V is a set of variables defined at I. Computing a program slice requires determining the nodes of possible dependence chains that end (backward slicing), or start (forward slicing) at the slicing criterion, respectively. The program slicing method is considered to be precise up to realizable program paths if the slice is computed upon realizable dependence chains.

2.1 Program Slicing via Token Propagation

The token propagation-based static program slicing method has been presented in [21]. The idea of the approach is to discover possible dependence chains by propagating tokens of the control flow graph starting from the slicing criterion. Tokens contain a token index corresponding to a defined variable (initially the variable of the slicing criterion) and a backtrack index used to control token propagations from procedure exit nodes (considering realizable program paths). Tokens are propagated along definition-clear paths wrt. variable corresponding to the token index; tokens propagated to affected nodes (containing use of the token index) causing these nodes marked as in the slice. Influenced definitions induce new token propagations from the affected nodes. A special $\emptyset$ backtrack index value is used to distinguish tokens having no previous "calling context", otherwise backtrack indices

correspond to variable identifiers.

The token propagation rules of forward data-flow slice computation (data-dependences are considered only) can be summarized as follows:

Rule 0. A token $RD_x^{\emptyset}$ is created for slicing criterion $C=<n, \{x\}>$, which is propagated to the successor node of n. Node n is marked as in the slice.

Rule 1. If a token RD_x^y is propagated to a node n that does not (re)define variable x, the token is propagated to the successor node(s) of n unchanged.

Rule 2. If a token RD_x^y is propagated to a node n that uses variable x, n is marked as in the slice. A new token RD_z^y is created for definition of variable z influenced by use of x, which is propagated to the successor node of n.

Rule 3. If a token RD_x^y is propagated to a call site, token RD_x^x is propagated to the entry node of the called procedure.

Rule 4. Any call site c that contains a token RD_x^y and exit node e (of the called procedure) that contains a token RD_z^x induce the propagation of a token RD_z^y from return site *returnSiteOf(c)*. Token $RD_z^{\emptyset}$ is propagated from an exit node to all return sites unchanged.

The token propagation stops when no more propagation is possible (a given token is propagated to a given node once), and the slice is given by the set of nodes marked as in the slice. Notice that a token RD_z^x propagated to a procedure exit node directly corresponds to procedure summary edge: $x \rightarrow z$ of Horwitz et al. [28]. (Summary edges represent the transitive dependences due to the procedure call.)

Control tokens are created at affected predicate nodes (using a special token index value C) and propagated along control edges to accommodate control dependences. Nodes reached by control tokens are marked as in the slice; definitions in control dependent nodes start new (data) token propagations. The rules of the full forward slicing are shown below:

Rule 5. If a token RD_x^y is propagated to a predicate node n that uses variable x, a new token RD_C^y is created and propagated to the nodes that are control dependent on n.

Rule 6. If a token RD_C^y is propagated to a predicate node n, token RD_C^y is propagated to the nodes that are control dependent on n.

Rule 7. If a token RD_C^y is propagated to a node n, n is marked as in the slice. A new token RD_z^y is created for definition of variable z, which is propagated to the successor node of n.

Rule 8. If a token RD_C^C is propagated to an entry node, token RD_C^C is propagated to all the nodes of the procedure (except entry and exit nodes).

Backward slicing can be obtained by reversing the token propagation rules of forward slicing, where tokens are propagated to predecessor nodes, backwards.

3 The Reason-why Algorithm

This section presents a method capable of reasoning about an arbitrarily selected element of the resulting slice, called the "*reason-why algorithm*". First, we restrict to forward data-flow slices; then, we extended to full forward slices. Reasoning about backward slices is just the dual of the presented method, which is hence omitted to save space. For clarity of the presentation we consider programs containing global and scalar variables. Local variables and parameter passing can be treated as described in [21].

3.1 Reasoning Data-flow Slices

We assume that we are given a slicing criterion $C=<n, \{x\}>$ for which the data-flow slice has been computed using the token propagation method. We also assume all the tokens propagated during slicing are available, and the resulting slice contains a node m to be explained; m contains a use of variable y and a token RD_y^z caused m to be marked as in the slice (Rule 2). To justify why m is included in the slice our goal is to present a definition-use chain from n to m – along with a potential execution trace that covers it. The pair $(n, RD_x^{\emptyset})$ will be referred to as the *source*; the pair (m, RD_y^z) is referred to as the *target*. We note that we provide one single, any of the possible definition-use chains between the source and the target, which is not necessarily the shortest one.

To our experiences providing a complete CFG path covering a definition-use chain contains too much detail (instructions) to overview by a human user; providing merely the nodes of the chain is not enough to see how this dependence chain can be covered by a potential program execution. The path to be constructed, called the "*reason-why path*", will hence be a definition-use chain augmented with procedure calls and returns (intraprocedural path segments between the use-definition nodes and the procedure boundaries are omitted).

To reveal a definition-use chain between n and m we trace back the token propagation performed during slicing. We start from target node m, and investigate the tokens propagated to the predecessor nodes. Based on this information we can deduce to the previously applied token propagation rule(s), and determine the node(s) from where the token propagated to m may have been originated. The predecessor node and the (possibly) new token propagated to the predecessor node become the new target. Then, we continue finding such predecessors as far as we reach the source. From procedure entry nodes we "return" to call sites, and from return sites we enter procedure exit nodes, respectively. The traversed definition-use chain nodes, as well as procedure call- and return sites are recorded; finally, this node sequence is reversed. We bypass recovering applications of Rule 1 (which propagates tokens unchanged to successors iteratively) by identifying reachable nodes along definition-clear paths backwards.

The construction of the reason-why path is performed in two passes: in Pass 1 we traverse intraprocedural-, summary- and call edges backwards (to callers), whereas in Pass 2 we traverse intraprocedural-, summary- and return edges (to

called procedures). As procedure summary edges – represented by exit node tokens in the called procedures – are available, we can cross procedure calls without ascending into the called procedures. Exploited summary edges are resolved in a subsequent step. Finally, the path is reversed to get a forward path. Note that using the two-pass method procedure calls and returns are correctly nested, i.e. the resulting reason-why path is realizable.

Pass 1

Pass 1 (as well as Pass 2) consists of a sequence of intra- and interprocedural path search steps. In the intraprocedural step our goal is to get to the entry node of the current procedure, whereas in the interprocedural step we select one of the potential callers of the current procedure from where the token propagation had been originated.

First, we consider the initial target: node m and token RD_y^z, where $z \neq \emptyset$. (If $z = \emptyset$, we skip Pass 1.) To determine the node from where RD_y^z had been propagated to m, we determine the set of nodes in the current procedure reachable along definition-clear path wrt. y backwards. The possible source(s) of RD_y^z among these nodes is either (a) the procedure entry node if $z = y$ and the entry node contains RD_y^y, (b) a node containing a definition of variable y, a use of a variable v, and a token RD_v^z (RD_y^z had been started by Rule 2), or (c) a return site of a called procedure P such that the related call site of P contains a token RD_v^z and there is a summary edge $v \rightarrow y$ (Rule 4 had been applied to RD_y^v in the called procedure's exit node). Note that as the backtrack index is not $\emptyset$, slicing criterion node n cannot be the source of RD_y^z. In either case, we record- and set the new node and the new token as the new *target*. In the case of (b) and (c), we continue searching for the next predecessor of the current target as far as we reach the entry. In the case of (c), we record the call- and the return site, as well as the summary edge used to cross the call (resolved later). To avoid infinite loop we traverse each node-token pair at most once, and use backtracking if necessary.

In the interprocedural step, we select one of the potential callers that resulted in the propagation of RD_y^y to the entry node. These call sites contain a token RD_y^v (Rule 3 had been applied). We select one of them, and apply the above intraprocedural path search for the new target (call site and RD_y^v) to get to the entry node of the caller procedure.

We continue the above procedure as far as any of the call sites contains a token $RD_y^\emptyset$, when we turn to Pass 2. In the presence of strongly-connected components (SCCs), we visit each call site and call site token at most once, which avoids infinite cycle.

As an example, let us consider the program shown in Figure 1. For slicing criterion C=($a2$, $\{x\}$), we obtain the data-flow slice: S=$\{a2, a4, b2, m6, c5\}$. (The related instructions are highlighted in boldface characters; tokens propagated during slicing are indicated next to the nodes in the figure). Assume that we choose node $c5$ to be explained.

In Pass 1, we start from target ($c5$, RD_y^y). After identifying the set of nodes reachable (backwards) along definition-clear paths wrt. y we find return site $c3$,

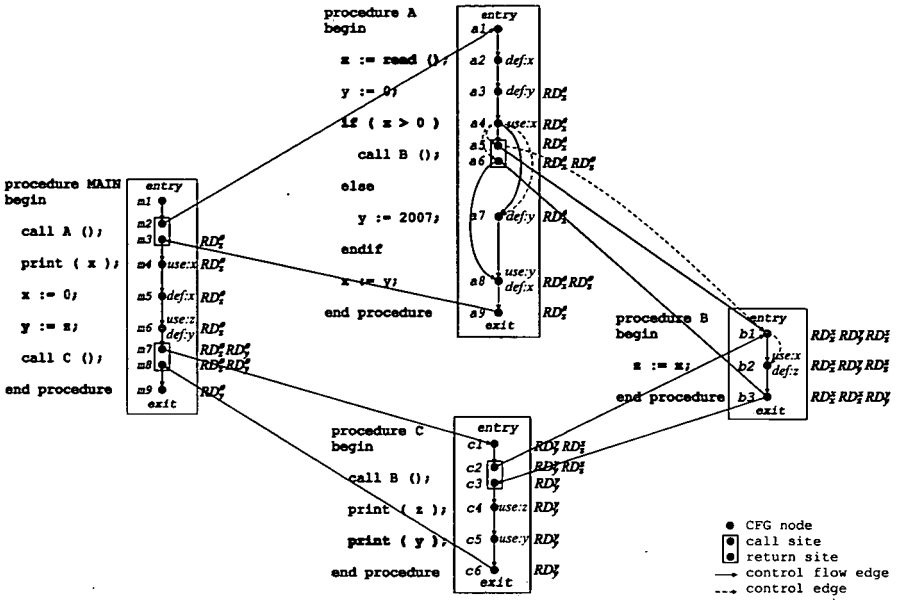

Figure 1: Example program graph and the tokens propagated during data-flow slicing

whose call site contains a token RD_y^y and the called procedure contains summary edge $y \rightarrow y$ (exit node token RD_y^y in procedure B; case c). The new target is set as node *c2* and token RD_y^y. In the next step, we reach procedure entry node *c1* (case a).

In the interprocedural step, we return to call site *m7*, as it contains a token $RD_y^\emptyset$, so we finish Pass 1. The reason-why path constructed during Pass 1 is shown below:

1. (c5, RD_y^y) -- use of y
2. (c3, RD_y^y) -- return from B, summary edge: $y \rightarrow y$
3. (c2, RD_y^y) -- call B
4. (c1, RD_y^y) -- entry C
5. (m7, $RD_y^\emptyset$) -- call C

Pass 2

During Pass 2 we traverse intraprocedural- and return edges, and trace back the propagation of $RD_y^\emptyset$ towards the slicing criterion.

The intraprocedural path search starts from a call site (following Pass 2), or from node m, respectively (m contains a token $RD_y^\emptyset$). The potential source of this token is a node reachable from the current target along definition clear-path wrt. y backwards, which is either (a) node n if $y = x$, (b) a node containing a definition

of variable y, a use of a variable v, and a token $RD_v^\emptyset$ (Rule 2), (c) a return site such that the related call site contains a token $RD_v^\emptyset$ and there is summary edge RD_v^y (Rule 4), or (d) a return site such that the called procedure's exit node contains the token $RD_y^\emptyset$ (Rule 4 is applied to a token with $\emptyset$ backtrack index). In the case of (a), we finish Pass 2; in the case of (b) or (c), we continue the intraprocedural search; in the case (d), we set the exit node of the called procedure and $RD_y^\emptyset$ as the new target (interprocedural step). We continue the above procedure as far as we reach n.

In the example, in Pass 2, we start from node *m7* and token $RD_y^\emptyset$. The only reachable node is node *m6*, which defines y, uses z, and contains a token $RD_z^\emptyset$ (case *b*). The new target is set as (*m6*, $RD_z^\emptyset$). In the next steps, we select return site *m3* and exit node *a9* of procedure A, which contains $RD_z^\emptyset$ (case *d*). The source of token $RD_z^\emptyset$ propagated to *a9* is return site *a6*, since there is a token $RD_x^\emptyset$ in *a5*, and the called procedure contains summary edge $x \rightarrow z$. From target (*a5*, $RD_x^\emptyset$) slicing criterion node *a2* is reachable, and token index x corresponds to the variable of the slicing criterion (case *a*), so Pass 2 finishes too.

The path constructed in Pass 2 is as follows:

```
6.  (m6, RD_z^∅) -- use of z, definition of y
7.  (m3, RD_z^∅) -- return from A
8.  (a9, RD_z^∅) -- exit A
9.  (a6, RD_z^∅) -- return from B, summary edge:  x → z
10. (a5, RD_x^∅) -- call B
11. (a2, RD_x^∅) -- definition of x
```

Resolving Summary Edges

The reason-why path potentially contains "jumps" from return- to call sites via summary edges that need to be resolved. It requires constructing a coverage path for a dependence-chain realizing the procedure summary. We iterate over each adjacent call- and return sites contained in the reason-why path, resolve them one-by-one, and insert the related summary edge coverage path into the original reason-why path between the related call- and return site pair.

The construction of the coverage path for a summary edge $v \rightarrow y$ is performed correspondingly to the intraprocedural path search applied in Pass 1: for a given call site c and return site r we construct a reason-why path from the exit node of the called procedure and token RD_y^v (target) to the entry node of the called procedure and token RD_v^v (source). Once this path has been constructed, it is inserted between the call- and return site pair.

Resolving a summary edge may introduce new summary edges (case *c*), which also need to be resolved, recursively. In the presence of SCCs, during resolving a summary edge the same summary edge could potentially be reused. As during resolving a summary edge there must exists a path that does not reuse itself (otherwise, it would mean an infinite loop in the code, so the summary edge would have

never been computed), excluding the reuse of the same summary edge currently being resolved, the infinite loop can be avoided.

By reversing the resulting path we obtain the required definition-use chain containing a proper sequence of call- and return sites.

Continuing with the example, the reason-why path contains two summary edges, at positions 2 and 9, which need to be resolved. The first summary edge $y \rightarrow y$ is resolved by starting from exit node *b3* in procedure B and token RD_y^y (target). Since the entry node is reachable from the exit, and the entry node contains RD_y^y (source), the path search finishes. The path to be inserted between positions 2 and 3 is as follows:

```
1.  (b3, RD_y^y) -- exit B
2.  (b1, RD_y^y) -- entry B
```

During resolving summary edge $x \rightarrow z$ of procedure B we have to traverse node *b2* as well, which results in the following path to be inserted between positions 9 and 10:

```
1.  (b3, RD_z^x) -- exit B
2.  (b2, RD_x^x) -- use of x, definition of z
3.  (b1, RD_x^x) -- entry B
```

The resulting reason-why path is then reversed. The reason-why path from *a2* to *c5* is shown below (only target token indices are indicated – corresponding to the most recently defined variable; procedure calls and returns are tabbed; comments are substituted by actual program instructions):

```
1.    a2, x  -- x := read()
2.    a5, x  -- call B ()
3.     b1, x -- entry B
4.     b2, x -- z := x
5.     b3, z -- exit B
6.    a6, z  -- return from B
7.    a9, z  -- exit A
8.   m3, z   -- return from A
9.   m6, z   -- y := z
10.  m7, y   -- call C ()
11.   c1, y  -- entry C
12.   c2, y  -- call B ()
13.    b1, y -- entry B
14.    b3, y -- exit B
15.   c3, y  -- return from B
16.   c5, y  -- print (y)
```

3.2 Reasoning Full Slices

In full slicing, data dependent predicates induce propagations of control tokens along control edges, which also need to be considered at constructing the reason-why path.

If target node m contains control token only, the initial target is of the form (m, RD_C^z). During the intraprocedural path search we determine the set of *controlling* nodes, i.e. the nodes from where there is a control edge to m. The possible source(s) of token RD_C^z among these nodes is either (a) a predicate node containing a use of a variable v and a token RD_v^z (Rule 5), (b) a predicate node containing RD_C^z (Rule 6), or (c) the procedure entry node if $z = C$ and the entry node contains RD_C^C (Rule 8). The new node and the new token are set as the new target. This intraprocedural search step is applied in passes 1 and 2 each time the origin of a control token needs to be determined.

Another change in reasoning full slices is that control tokens induce data-tokens at definition nodes (Rule 7); hence, at determining the possible sources of a data token RD_y^z, nodes containing definition of variable y and token RD_C^z need to be investigated as well. If it holds for some node, this node and RD_C^z are also a potential new target during the intraprocedural path search.

When the target token is a control token, the interprocedural step in Pass 1 requires determining the set of call sites containing control token. In Pass 2, as no control token can be propagated to a procedure exit node, the interprocedural traversal is unchanged.

Using the above extensions, a reason-why path can also be calculated for elements of full slices.

4 Related Work

Various algorithms for calculating interprocedural slices exist. The first method published by Weiser [35] is not context-sensitive. There are studies [1, 32, 6, 31] investigating whether considering calling-context has significant affect on the size of the slices. It may occur that inaccurate slices due to following non-realizable paths are several times larger than precise ones – what is more, the computation of these extra large slices may take more time.

There are a number of context-sensitive static slicing methods. Most of them are based on system dependence graphs published first by Horwitz et al. [28]. By computing transitive dependences due to procedure calls (summary edges), slicing is reduced to a graph reachability problem. Agrawal and Guo [1] have presented an explicitly context-sensitive slicing method over the SDG (without summary edges), in which the call stack is maintained during the propagation. Krinke [32] presented a corrected explicitly context-sensitive algorithm. Atkinson and Griswold [3] used CFGs and the invocation graph approach [13] for context-sensitive slicing. Liang and Harrold [33] proposed a precise slice computation method also based on data-flow information propagation over the CFG.

To our knowledge no reasoning technique has been proposed to justify slice elements computed by these methods.

Hajnal and Forgács [21] presented a context-sensitive static slicing method which combines the demand-driven nature of the CFG-based slicing and the efficiency of the SDG-based slicing, using token propagation. The reason-why algorithm proposed in this paper makes it possible to justify slice elements computed that method but also applicable to reason about slice elements computed by any other technique which is at least as precise as our token propagation-based method. For example, our method can be applied for SDG-based slicing considered the most wide-spread method nowadays

Chopping [29] is a variant of program slicing capable of revealing statements involved in a transitive dependence from one specific statement (source criterion) to another one (target criterion). A chop is basically the intersection of the forward slice of the forward criterion and the backward slice of the backward criterion, which provides a more focused approach to investigating how one statement affects the other. Considering a chop, which gives a set of nodes composed of (all) the dependence chains between the source and the target, it can be still very difficult to construct a dependence chain from source to target – and, if given that, an appropriate calling sequence that covers these nodes. The solution proposed in this paper answers both questions. We are aware of no other similar techniques for this problem.

5 Conclusions

To our knowledge no automated reasoning technique about the computed slice elements has been proposed in the literature so far. Without such a tool verification or understanding of the resulting program slices requires considerable expertise and time. This paper proposes a solution to "explain" slice elements by computing a specific dependence chain from the slicing criterion to the chosen slice element. This definition-use chain, augmented with control information, is more easily overviewed or analyzed by a human user.

We implemented the presented reason-why algorithm in the Java programming language and integrated with the slicing tool presented in [21]. We carried out several experiments on the same COBOL systems and slices computed in programs of different sizes. The results showed that in all the cases the slice computation time dominates the time of the reason-why path computation (it took only a few seconds in the worst case). It is because the reason-why algorithm only reads the available token information and performs no compute-intensive operations (such as slicing). Note that slice computation has to be performed once; then several reasoning tasks can be initiated on the resulting slice elements.

In the presented method, the dependence chain is determined arbitrarily – tracing back any of the possible token propagations performed during previous slicing. Shorter chains are however easier to understand, therefore we plan to investigate how to provide shorter paths from source to target (e.g., by also introducing a

kind of distance information from source into tokens). As the number of tokens may be huge in the case of large programs, it is an interesting question how the increased memory requirements and its maintenance cost affect the overall slicing performance, and in what extent the reason-why chains can be shortened. Also, to our experiences data-dependences are easier to follow mentally, hence, we should be able to give option for selecting data-dependences (priorize or let the user decide interactively) where both types of dependences arise. It would also be worth investigating that if the provided path has been found infeasible by the user, how the algorithm can search for alternative path. The latter issues imply further possibilities for improvement: how to visualize or represent the reason-why path (which is currently plain XML), and how to make the path search interactive, respectively. These serve as the basis of our future work.

References

[1] Agrawal, G., and Guo, L. Evaluating explicitly context-sensitive program slicing. *Proceedings of the 2001 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis For Software Tools and Engineering*, pages 6–12, 2001.

[2] Agrawal, H., and Horgan, J. Dynamic program slicing. *ACM SIGPLAN Conference on Programming Language Design and Implementation* (PLDI). ACM, New York, USA, pages 246–256, 1990.

[3] Atkinson, D.C., and Griswold, W.G. The design of whole-program analysis tools. *Proceedings of the 18th International Conference on Software Engineering*, pages 16–27, 1996.

[4] Binkley, D. Semantics guided regression test cost reduction. *IEEE Transactions on Software Engineering*, 23(8): 498–516, 1997.

[5] Binkley, D. The application of program slicing to regression testing. *Information and Software Technology Special Issue on Program Slicing*, 40(11–12): 583–594, 1998.

[6] Binkley, D., and Harman, M. A large-scale empirical study of forward and backward static slice size and context sensitivity. *Proceedings of the International Conference on Software Maintenance*, pages 44–53, 2003.

[7] Binkley, D., Horwitz, S., and Reps, T. Program integration for languages with procedure calls. *Transactions on Programming Languages and Systems*, 4(1): 3–35, 1995.

[8] Canfora, G., Cimitile, A., and De Lucia, A. Conditioned program slicing. *Information and Software Technology Special Issue on Program Slicing*, 40(11–12): 595–607, 1998.

[9] Canfora, G., Cimitile, A., and Munro, M. RE2: Reverse engineering and reuse reengineering. *Journal of Software Maintenance: Research and Practice*, 6(2): 53–72, 1994.

[10] Canfora, G., Cimitile, A., De Lucia, A., and Di Lucca, G.A. Software salvaging based on conditions. *Proceedings of the International Conference on Software Maintenance*, pages 424–433, 1994.

[11] Cimitile, A., De Lucia, A., and Munro, M. A specification driven slicing process for identifying reusable functions. *Journal of Software Maintenance: Research and Practice*, 8(3): 145–178, 1996.

[12] De Lucia, A., Fasolino, A.R., and Munro, M. Understanding function behaviours through program slicing. *Proceedings of the 4th IEEE Workshop on Program Comprehension*, pages 9–18, 1996.

[13] Emami, M., Ghiwya, R., and Hendren, L.J. Context-sensitive interprocedural points-to analysis in the presence of function pointers. *Proceedings of the ACM SIGPLAN 1994 Conference on Programming Language Design and Implementation*, pages 20–24, 1994.

[14] Forgács, I., Takács, É., and Hajnal, Á. Regression slicing and its use in regression testing. *Proceedings of IEEE International Computer Software and Applications Conference*, pages 464–469, 1998.

[15] Gallagher, K.B. Evaluating the surgeon's assistant: Results of a pilot study. *Proceedings of the Conference on Software Maintenance*, pages 236–244, 1992.

[16] Gallagher, K.B., and Lyle, J.R. Using program slicing in software maintenance. *IEEE Transactions on Software Engineering*, 17(8): 751–761, 1991.

[17] Griswold, W.G. Making slicing practical: the final mile. *Proceedings of the 2001 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, page 1, 2001.

[18] Gupta, R., Harrold, M.J., and Soffa, M.L. An approach to regression testing using slicing. *Proceedings of the Conference on Software Maintenance*, pages 299–308, 1992.

[19] Gupta, R., Soffa, M.L., and Howard, J. Hybrid Slicing: Integrating Dynamic Information with Static Analysis. *ACM Transactions on Software Engineering and Methodology*, 6(4): 370–397, 1997.

[20] Gyimóthy, T., Beszédes, Á., and Forgács, I. An efficient relevant slicing method for debugging. *Lecture Notes in Computer Science*, pages 303–321, 1999.

[21] Hajnal, Á., and Forgács, I. A demand-driven approach to slicing legacy COBOL systems. *Journal of Software: Evolution and Process*, 24(1): 67–82, 2012.

[22] Harman, M., and Danicic, S. Amorphous program slicing. *Proceedings of the 5th International Workshop on Program Comprehension*, pages 70–79, 1997.

[23] Harman, M., and Danicic, S. Using program slicing to simplify testing. *Software Testing, Verification and Reliability*, 5(3): 143–162, 1995.

[24] Harman, M., Hierons, R.M., Fox, C., Danicic, S., and Howroyd, J. Pre/Post conditioned slicing. *Proceedings of the IEEE International Conference on Software Maintenance*, pages 138–147, 2001.

[25] Hierons, R., Harman, M., and Danicic, S. Using program slicing to assist in the detection of equivalent mutants. *Software Testing, Verification and Reliability*, 9(4): 233–262, 1999.

[26] Hierons, R., Harman, M., Fox, C., Ouarbya, L., and Daoudi, M. Conditioned slicing supports partition testing. *Software Testing, Verification and Reliability*, 12(1): 23–28, 2002.

[27] Horwitz, S., Prins, J., and Reps, T. Integrating non-interfering versions of programs. *Transactions on Programming Languages and Systems*, 11(3): 345–387, 1989.

[28] Horwitz, S., Reps, T., and Binkley, D. Interprocedural slicing using dependence graphs. *ACM Transactions on Programming Languages and Systems*, 12(1): 26–60, 1990.

[29] Jackson, D. and Rollins, E.J. A new model of program dependences for reverse engineering. *Proceedings of the Second ACM SIGSOFT Symposium on Foundations of Software Engineering*, pages 2–10, 1994.

[30] Korel, B., and Laski, J. Dynamic program slicing. *Information Processing Letters*, 29(3):155–163, 1988.

[31] Krinke, J. Effects of context on program slicing. *Journal of Systems and Software*, 79(9): 1249–1260, 2006.

[32] Krinke, J. Evaluating context-sensitive slicing and chopping. *Proceedings of the International Conference on Software Maintenance*, pages 22–31, 2002.

[33] Liang, D., and Harrold, M. J. Reuse-Driven Interprocedural Slicing in the Presence of Pointers and Recursion. *Proceedings of the IEEE International Conference on Software Maintenance*, pages 421–430, 1999.

[34] Venkatesh, G.A. The semantic approach to program slicing. *Proceedings of the ACM SIGPLAN 1991 Conference on Programming Language Design and Implementation*, pages 107–119, 1991.

[35] Weiser, M. Program slicing. *IEEE Trans. Software Eng.*, 10(4): 352–357, 1984.

Received 10th April 2012

Acta Cybernetica **20** (2012) 499–513.

New Descriptions of the Lovász Number, and the Weak Sandwich Theorem

Miklós Ujvári*

Abstract

In 1979, L. Lovász introduced the concept of an orthonormal representation of a graph, and also a related value, now popularly known as the Lovász number of the graph. One of the remarkable properties of the Lovász number is that it lies sandwiched between the stability number of the graph and the chromatic number of the complementary graph. This fact is called the sandwich theorem.

In this paper, using new descriptions of the Lovász number and linear algebraic lemmas we give three proofs for a weaker version of the sandwich theorem.

Keywords: Lovász number, weak sandwich theorem

1 Introduction

From the several remarkable properties of the Lovász number of a graph we mention here only the sandwich theorem: the Lovász number lies 'sandwiched' between the stability number, and the chromatic number of the complementary graph. A weaker form of this sandwich theorem will be derived here using new descriptions of the Lovász number. This weak sandwich theorem is the immediate consequence of the sandwich theorem, Brooks' theorem (concerning an upper bound on the chromatic number), and the counterpart of Brooks' theorem (concerning a lower bound on the stability number). In this paper our aim is to give more direct proofs.

We begin this paper with stating the above-mentioned results. First we fix some notation. Let $n \in \mathcal{N}$, and let $G = (V(G), E(G))$ be an undirected graph, with vertex set $V(G) = \{1, \ldots, n\}$, and with edge set $E(G) \subseteq \{\{i,j\} : i \neq j\}$. The complementary graph will be denoted by $\overline{G}$. Thus $\overline{G} = (V(\overline{G}), E(\overline{G}))$ where $V(\overline{G}) = V(G)$ and $E(\overline{G}) = \{\{i,j\} \subseteq V(G) : i \neq j,\ \{i,j\} \notin E(G)\}$.

Let us define an *orthonormal representation* of the graph G (shortly, o.r. of G) as a system of vectors $a_1, \ldots, a_n \in \mathcal{R}^m$ for some $m \in \mathcal{N}$, satisfying

$$a_i^T a_i = 1 \ (i = 1, \ldots, n),\ a_i^T a_j = 0 \ (\{i,j\} \in E(\overline{G})).$$

*H-2600 Vác, Szent János utca 1., Hungary. E-mail: ujvarim@cs.elte.hu

In the seminal work [8] L. Lovász introduced the following number, $\vartheta(G)$, now popularly known as the *Lovász number* of the graph G ([7]):

$$\vartheta(G) := \inf\left\{\max_{1\le i\le n}\frac{1}{(a_i a_i^T)_{11}} : a_1,\dots,a_n \text{ o.r. of } G\right\}.$$

(Here $(a_i a_i^T)_{11}$ denotes the upper left corner element of the matrix $a_i a_i^T$, that is the square of the first element of the vector a_i, and though not emphasized in the definition of $\vartheta(G)$, we suppose that $(a_i a_i^T)_{11} \neq 0$ for all $i \in V(G)$.)

By Lemma 3 in [8], the Lovász number $\vartheta(G)$ is an upper bound for the stability number $\alpha(G)$, the maximum cardinality of the (so-called stable) sets $S \subseteq V(G)$ such that $\{i,j\} \subseteq S$ implies $\{i,j\} \notin E(G)$. Moreover, by Theorem 11 in [8] if there exists an orthonormal representation of the graph G with vectors a_i in $\mathcal{R}^m$ then $\vartheta(G) \le m$. Specially, $\vartheta(G)$ is at most the chromatic number of the complementary graph, $\chi(\overline{G})$, where the chromatic number of a graph is the minimal number of stable sets covering the vertex set of the graph. Hence (see [8])

$$\alpha(G) \le \vartheta(G) \le \chi(\overline{G}),$$

a fact known as the sandwich theorem (see [7]).

The Lovász number can also be defined via orthonormal representations of the complementary graph: it is shown in [8] that $\vartheta(G) = \vartheta'(G)$ where the number $\vartheta'(G)$ is defined as

$$\vartheta'(G) := \sup\left\{\sum_{i=1}^{n}(b_i b_i^T)_{11} : b_1,\dots,b_n \text{ o.r. of } \overline{G}\right\}.$$

(We remark that here the values $(b_i b_i^T)_{11}$ are allowed to be zero.) The proof of the equality $\vartheta(G) = \vartheta'(G)$ relies on strong duality between Slater-regular primal-dual semidefinite programs equivalent with the programs defining $\vartheta(G)$ and $\vartheta'(G)$, respectively. (See [8], [10] or [15] for the equivalency results; and, for example, [16], [17] for the duality results.) As a consequence of the sandwich theorem and the equality between the values $\vartheta(G)$ and $\vartheta'(G)$ we have

$$\alpha(G) \le \vartheta'(G) \le \chi(\overline{G}),$$

a fact that can also be derived easily from the definition of $\vartheta'(G)$.

For $i \in V(G)$ let $N(i)$ denote the set of vertices $j \in V(G)$ such that $\{i,j\} \in E(G)$. Let us denote by d_i the cardinality of the set $N(i)$, and let $d_{\max}$ denote the maximum of the values d_i ($i \in V(G)$). We define similarly $\overline{N}(i)$, $\overline{d}_i$ and $\overline{d}_{\max}$ for the complementary graph $\overline{G}$ instead of G.

The following theorem is well-known (see for example [9]):

Theorem 1.1. *(Brooks) For any graph G, the chromatic number $\chi(G)$ is at most $d_{\max}+1$, with equality for a connected graph G if and only if the graph is a clique or an odd cycle.*

As a corollary of Theorem 1.1 and the sandwich theorem we obtain

Corollary 1.1. *The value $\vartheta(G)$ ($\vartheta'(G)$ also) is at most $\bar{d}_{\max} + 1$.*

The counterpart of the Brooks' theorem can be found for example in [1]. For further lower bounds on the stability number, see [3], [18].

Theorem 1.2. *(Caro-Wei) For any graph G, the stability number $\alpha(G)$ is at least $\sum_{i \in V(G)} 1/(d_i + 1)$, with equality if and only if the graph G is the disjoint union of cliques.*

Similarly as in the case of Theorem 1.1 we have the following corollary:

Corollary 1.2. *The value $\vartheta'(G)$ ($\vartheta(G)$ also) is at least $\sum_{i \in V(G)} 1/(d_i + 1)$.*

We will call the results described in Corollaries 1.1 and 1.2 together the weak sandwich theorem. In Sections 2 and 3 we give two proofs for this theorem using linear algebraic lemmas and new descriptions of the Lovász number. In Section 4 we present a new, elementary graph theoretical proof for Theorem 1.2 (which is a derandomization of the original proof) thus obtaining a third proof for the weak sandwich theorem.

2 First proof for the weak sandwich theorem

In the first proof of the weak sandwich theorem we will need the following lemma, implicit in the proof of Theorem 3 in [8]:

Lemma 2.1. *Let* PSD *denote the set of n by n real symmetric positive semidefinite matrices. Let P denote the following set of matrices:*

$$P := \left\{ \left(\frac{a_i^T a_j}{e_1^T a_i \cdot e_1^T a_j} - 1 \right) \;\middle|\; \begin{array}{l} m \in \mathcal{N};\ a_i \in \mathcal{R}^m\ (1 \le i \le n); \\ a_i^T a_i = 1\ (1 \le i \le n) \end{array} \right\}.$$

Then PSD $= P$. *(Here e_1 denotes the first column vector of the identity matrix I. Though not emphasized in the definition of the set P, we suppose that the vectors a_i have nonzero first coordinates, that is $e_1^T a_i \neq 0$ for $i = 1, \ldots, n$.)*

Proof. First we will prove the inclusion $P \subseteq$ PSD. Let $a_1, \ldots, a_n$ be unit vectors. Then the vectors $a_i \cdot (e_1^T a_i)^{-1}$ can be written as $(1, x_i^T)^T$ with appropriate vectors x_i. We have

$$\left(\frac{a_i^T a_j}{e_1^T a_i \cdot e_1^T a_j} - 1 \right) = (x_i^T x_j) \in \text{PSD}.$$

Thus the elements of the set P are positive semidefinite.

To prove the reverse inclusion PSD $\subseteq P$, let X be a positive semidefinite matrix. Then, there exist vectors x_i such that $X = (x_i^T x_j)$. Let $a_i := \lambda_i (1, x_i^T)^T$ where

the constants λ_i are chosen appropriately so that $a_i^T a_i = 1$ holds. With these definitions we have

$$X = (x_i^T x_j) = ((1, x_i^T)(1, x_j^T)^T - 1) = \left(\frac{a_i^T a_j}{e_1^T a_i \cdot e_1^T a_j} - 1 \right).$$

Thus $X \in P$, which was to be shown. □

From Lemma 2.1 follows immediately that the program defining $\vartheta(G)$ and the following program are equivalent:

$$\inf \max_{1 \le i \le n} x_{ii} + 1,\ x_{ij} = -1\ (\{i,j\} \in E(\overline{G})),\ X \in \text{PSD}. \tag{1}$$

(We remark that program (1) in an equivalent form was studied previously by Meurdesoif, see program $(\mathcal{P}_\mathcal{L})$ in [11].) We can see

Theorem 2.1. *The optimal value of program (1) is equal to $\vartheta(G)$, and it is attained.* □

Now, let X be the following matrix:

$$X := (x_{ij}),\ \text{where } x_{ij} := \begin{cases} \overline{d}_i, & \text{if } i = j, \\ 0, & \text{if } \{i,j\} \in E(G), \\ -1, & \text{if } \{i,j\} \in E(\overline{G}). \end{cases}$$

Then $x_{ii} \ge \sum_{i \ne j} |x_{ij}|$ holds for $1 \le i \le n$, so the matrix X is positive semidefinite by the Gerschgorin's disc theorem, see [14]. (We can also use the fact that X is the Laplacian matrix corresponding to the adjacency matrix of $\overline{G}$, see [16] for another application of the Laplacian matrix.) Moreover, the matrix X is a feasible solution of program (1), with corresponding value $\overline{d}_{\max} + 1$. Thus we have $\vartheta(G) \le \overline{d}_{\max} + 1$, and Corollary 1.1 is proved. □

Similarly on the dual side we can apply the variable transformation described in Lemma 2.1 to the program defining $\vartheta'(G)$. This way we obtain the following program:

$$\sup \sum_{i=1}^{n} \frac{1}{y_{ii} + 1},\ y_{ij} = -1\ (\{i,j\} \in E(G)),\ Y \in \text{PSD}. \tag{2}$$

The optimal value of program (2) is a lower bound of $\vartheta'(G)$, as when writing program (2) we considered only the representations (b_i) where the vectors b_i had nonzero first coordinates. From these considerations Corollary 1.2 follows similarly as in the case above Corollary 1.1. □

We remark that the program defining $\vartheta'(G)$, and the program (2) are not equivalent generally. In fact, let G_0 be the cherry graph:

$$G_0 := (\{1,2,3\}, \{\{1,2\}, \{1,3\}\}).$$

Then $\vartheta'(G_0) = 2$ by the sandwich theorem, but the program (2) has no feasible solution with corresponding value 2. Otherwise there would exist

$$Y = \begin{pmatrix} x & -1 & -1 \\ -1 & y & w \\ -1 & w & z \end{pmatrix} \in \text{PSD}$$

such that

$$\frac{1}{x+1} + \frac{1}{y+1} + \frac{1}{z+1} = 2.$$

All the principal submatrices of the positive semidefinite matrix Y have nonnegative determinants, see [14]. Hence, $xy, xz \geq 1$, $x, y, z > 0$, and

$$x = \frac{1 - yz}{2yz + y + z}$$

would hold. From these relations $(1 - yz)y \geq 2yz + y + z$, that is $0 \geq z(y+1)^2$ would follow, which is a contradiction. This contradiction shows that there exist graphs such that in every optimal orthonormal representation (b_i) there exist at least one vector b_i with zero first coordinate.

In the next two propositions we describe two lower bounds for the optimal value of program (2).

Proposition 2.1. *The optimal value of program (2) is at least* $n/\vartheta(\overline{G})$.

Proof. Let $\varepsilon \geq 0$, and let $X = X(\varepsilon) \in \mathcal{R}^{n \times n}$ be a feasible solution of program (1) with $\overline{G}$ instead of G, such that

$$\max_{1 \leq i \leq n} x_{ii} + 1 \leq \vartheta(\overline{G}) + \varepsilon.$$

Then the matrix X is a feasible solution of program (2), and

$$n = \sum_{i=1}^{n} \frac{1}{x_{ii}+1} \cdot (x_{ii}+1) \leq \max_{1 \leq i \leq n} (x_{ii}+1) \cdot \sum_{i=1}^{n} \frac{1}{x_{ii}+1}.$$

We can see that $n/(\vartheta(\overline{G}) + \varepsilon)$ is a lower bound for the optimal value of program (2), and $\varepsilon \to 0$ (or $\varepsilon = 0$) gives the statement. □

Proposition 2.2. *The optimal value of program (2) is at least* $\alpha(G)$.

Proof. Let $S \subseteq V(G)$ be a stable set with cardinality $\alpha(G)$, and let $\varepsilon > 0$. Let us define the matrix $Y = Y(\varepsilon) \in \mathcal{R}^{n \times n}$ the following way:

$$Y := (y_{ij}), \text{ where } y_{ij} := \begin{cases} \varepsilon, & \text{if } i = j \in S, \\ 0, & \text{if } i, j \in S,\ i \neq j, \\ \lambda, & \text{if } i = j \notin S, \\ -1 & \text{otherwise.} \end{cases}$$

Here let $\lambda = \lambda(\varepsilon) \in \mathcal{R}$ be the minimum number such that Y is positive semidefinite, that is

$$\lambda := \left(1 + \frac{\alpha(G)}{\varepsilon}\right) \cdot (n - \alpha(G)) - 1.$$

(Schur complements [12] can be used to determine λ.) Then Y is a feasible solution of program (2). It can be easily seen that the corresponding value increases to $\alpha(G)$ while $\varepsilon > 0$ decreases to 0. □

From Propositions 2.1 and 2.2 equality between the optimal value of program (2) and $\vartheta'(G)$ follows:

- for vertex-transitive graphs where the lower bound $n/\vartheta(\overline{G})$ (Proposition 2.1) and the upper bound $\vartheta(G)$ are equal, see Theorem 8 in [8];
- for perfect graphs where the lower bound $\alpha(G)$ (Proposition 2.2) and the upper bound $\vartheta(G)$ are equal by the sandwich theorem and the perfect graph theorem [9].

Note that in the case of vertex-transitive graphs the optimal value of program (2) is attained, while in the case of perfect graphs non-attainment is possible.

Equality holds in the general case as well:

Theorem 2.2. *The optimal value of program (2) and $\vartheta'(G)$ are equal.*

Proof. Let us denote by $\mathrm{TH}(G)$ the set of vectors $x = (x_i) \in \mathcal{R}^n$ for which there exists a matrix $Z = (z_{ij}) \in \mathcal{R}^{n \times n}$ satisfying both

$$\begin{pmatrix} 1 & x^T \\ x & Z \end{pmatrix} \in \mathrm{PSD}$$

and

$$z_{ii} = x_i \ (i = 1, \ldots, n), \ z_{ij} = 0 \ (\{i, j\} \in E(G)).$$

It can be shown (see Corollary 29 in [7]) that $\mathrm{TH}(G)$ can be described alternatively as the set of vectors $x = (x_i) \in \mathcal{R}^n$ such that

$$x_i = (e_1^T b_i)^2 \ (i = 1, \ldots, n)$$

for some (b_i) orthonormal representation of the complementary graph $\overline{G}$.

Let $\mathrm{TH}_+(G)$ denote the set of positive vectors of $\mathrm{TH}(G)$. Then $\mathrm{TH}_+(G)$ is a convex set (as $\mathrm{TH}(G)$ is a convex set), and it is nonempty (as every graph can be represented by vectors with nonzero first elements). From this observation easily follows that

$$\mathrm{TH}_+(G) \subseteq \mathrm{TH}(G) \subseteq \mathrm{cl}\,\mathrm{TH}_+(G),$$

where cl denotes closure. Consequently we obtain the same value optimizing any linear function over $\mathrm{TH}(G)$ and $\mathrm{TH}_+(G)$; which is exactly the statement, for the linear function $(x_1, \ldots, x_n) \mapsto \sum_i x_i$. □

3 Second proof for the weak sandwich theorem

In this section we give an alternative proof for the weak sandwich theorem using a completely different technique than the one used in the previous section.

Let $\sigma(n)$ denote the number of integers s in the range $0 < s < n$ such that $s \equiv 0, 1, 2$ or $4 \pmod 8$. For small values of n, the value $\sigma(n)$ can be read out from the following table:

Table 1: The values $\sigma(n)$ for $n = 1, \ldots, 16$.

n	1	2	3,4	5,6,7,8
$\sigma(n)$	0	1	2	3
n	9	10	11,12	13,14,15,16
$\sigma(n)$	4	5	6	7

The table can be continued in a similar manner for larger values of n. With this notation the following combinatorial lemma holds:

Lemma 3.1. *If $n \geq 2$ then there exist n of $\sigma(n)$-letter words made up from the letters a, b, c, d such that the number of letter-pairs (a, b) and (c, d) on the same position in any two of the words is altogether odd. (For example in the words "aa" and "cb" there is only one such letter-pair: (a, b), on the second position.)*

Proof. For the values $2 \leq n \leq 9$ the following word-sets have the desired property:

$$\begin{aligned}
n = 2,\ \sigma(n) = 1: &\quad a, b \\
n = 3 \text{ or } 4,\ \sigma(n) = 2: &\quad \text{any } n \text{ words from the word-set } aa, cb, ba, db \\
n = 5, 6, 7 \text{ or } 8,\ \sigma(n) = 3: &\quad \text{any } n \text{ words from the word-set} \\
&\quad aaa, ccb, cba, cdb, baa, dab, dbc, dbd \\
n = 9,\ \sigma(n) = 4: &\quad aaaa, accb, acba, acdb, abaa \\
&\quad adab, adbc, cdbd, ddbd.
\end{aligned}$$

For larger values of n we can use the following induction argument. Let us denote by $S_1, \ldots, S_9$ the words defined above in the case $n = 9$. Suppose that for some n we have appropriate $\sigma(n)$-letter words $T_1, \ldots, T_n$. Then the word-set

$$S_1 \& T_1, \ldots, S_9 \& T_1,\ bdbd \& T_2, \ldots, bdbd \& T_n,$$

where & denotes concatenation, is made up of $n + 8$ of $(\sigma(n) + 4)$-letter words, and it has the desired property, too. Thus the statement in the lemma is dealt with for all the values of n. □

Now, let

$$A := \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, B := \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}, C := \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, D := \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

These matrices are orthogonal, furthermore from the matrix set

$$A^TB, A^TC, A^TD, B^TC, B^TD, C^TD$$

the matrices A^TB and C^TD are skew-symmetric, the others are symmetric. Given a word made up of the letters a, b, c, and d we can define a matrix by Kronecker-multiplying the corresponding matrices: for example the word "dbc" is transformed into the 8 by 8 matrix $D \otimes B \otimes C$ where $\otimes$ denotes Kronecker product. (The *Kronecker product* of two matrices $X = (x_{ij}), Y$ is the block matrix $X \otimes Y := (x_{ij} \cdot Y)$, see for example [12].) The matrices obtained this way are orthogonal, as they are the Kronecker products of orthogonal matrices.

Using this construction, from Lemma 3.1 immediately follows

Lemma 3.2. *If $m = 2^{\sigma(n)}$ then there exist m by m orthogonal matrices $C_1, \ldots, C_n$ such that for each $i \neq j$, the matrix $C_i^TC_j$ is skew-symmetric.*

Proof. Transform a word-set with the properties described in Lemma 3.1 into a matrix-set using the construction described before Lemma 3.2. We claim that this matrix-set meets the requirements. For example consider the matrix-set

$$A \otimes A, C \otimes B, B \otimes A, D \otimes B.$$

As we have noted already, these m by m matrices are orthogonal. On the other hand,

$$\begin{aligned}(A \otimes A)^T \cdot (C \otimes B) &= (A^T \otimes A^T) \cdot (C \otimes B) = (A^TC) \otimes (A^TB) = \\ &= (C^TA) \otimes (-B^TA) = -(C^T \otimes B^T) \cdot (A \otimes A) = -(C \otimes B)^T \cdot (A \otimes A),\end{aligned}$$

and similarly for the other matrix-pairs:

$$(A \otimes A)^T \cdot (B \otimes A) = -(B \otimes A)^T \cdot (A \otimes A) \ldots \text{ etc.}$$

In the general case similar argument can be applied, so the lemma is proved. □

We remark that in [13] Radon proved that there exist m by m orthogonal matrices $\tilde{C}_1, \ldots, \tilde{C}_n$ such that for each $i \neq j$ the matrix $\tilde{C}_i^T\tilde{C}_j$ is skew-symmetric if and only if $m \equiv 0 \pmod{2^{\sigma(n)}}$ (see also [6], [12]). The "if" part is an easy consequence of Lemma 3.2: just Kronecker-premultiply the C_i matrices with an identity matrix of appropriate dimension. For a similar proof of this part of Radon's theorem, see [4].

We will need one further lemma, concerning new descriptions of the Lovász number. The idea is to represent the graph G with matrices instead of vectors. Let us define a *matrix orthonormal representation* of the graph G (shortly, m.o.r. of G) as a block matrix $(A_1, \ldots, A_n) \in (\mathcal{R}^{\ell \times s})^n$ for some $\ell, s \in \mathcal{N}$, satisfying

$$A_i^TA_i = I \ (i = 1, \ldots, n), \ A_i^TA_j = 0 \ (\{i,j\} \in E(\overline{G})).$$

Then, let us define

$$\hat{\vartheta}(G) := \inf \left\{ \max_{1 \leq i \leq n} \frac{1}{(A_iA_i^T)_{11}} : (A_1, \ldots, A_n) \text{ m.o.r. of } G \right\}$$

and

$$\check{\vartheta}(G) := \sup\left\{\sum_{i=1}^{n}(B_iB_i^T)_{11} : (B_1,\ldots,B_n) \text{ m.o.r. of } \overline{G}\right\}.$$

It is obvious that $\hat{\vartheta}(G) \leq \vartheta(G)$ and $\vartheta'(G) \leq \check{\vartheta}(G)$. Equalities here follow from Lemma 3.3.

Lemma 3.3. *With the above definitions the inequality $\check{\vartheta}(G) \leq \hat{\vartheta}(G)$ holds.*

Proof. We adapt the proof of Lemma 4 in [8].

Let $(A_1,\ldots,A_n)$ and $(B_1,\ldots,B_n)$ be matrix orthonormal representations of G and $\overline{G}$, respectively. Then,

$$(A_i \otimes B_i)^T \cdot (A_j \otimes B_j) = (A_i^T A_j) \otimes (B_i^T B_j) = 0 \ (1 \leq i,j \leq n;\ i \neq j).$$

Thus the column vectors of the matrices $A_i \otimes B_i$ $(1 \leq i \leq n)$ altogether form an orthonormal system. Hence,

$$\sum_{i=1}^{n}((A_i \otimes B_i)(A_i \otimes B_i)^T)_{11} \leq 1,$$

which can be rewritten as

$$\sum_{i=1}^{n}(A_iA_i^T)_{11} \cdot (B_iB_i^T)_{11} \leq 1.$$

From this inequality

$$\min_{1\leq i\leq n}(A_iA_i^T)_{11} \cdot \sum_{i=1}^{n}(B_iB_i^T)_{11} \leq 1$$

follows, and so

$$\sum_{i=1}^{n}(B_iB_i^T)_{11} \leq \max_{1\leq i\leq n}\frac{1}{(A_iA_i^T)_{11}}$$

holds. We can see that $\check{\vartheta}(G) \leq \hat{\vartheta}(G)$, and the proof of the lemma is finished. □

Lemma 3.3, together with the equality $\vartheta(G) = \vartheta'(G)$, gives

Theorem 3.1. *The values $\hat{\vartheta}(G)$ and $\check{\vartheta}(G)$ are equal to $\vartheta(G)$ ($\vartheta'(G)$ also), and are attained.* □

The weak sandwich theorem is an easy consequence of Lemmas 3.2 and 3.3. Let $C_1,\ldots,C_n$ be m by m orthogonal matrices with the property described in Lemma 3.2. Let us define the matrices $A_1,\ldots,A_n$ the following way: the matrix will be $(1+\overline{e})m$ by m where $\overline{e}$ denotes the cardinality of $E(\overline{G})$. The first m by m block in A_i is α_iC_i where

$$\alpha_i := \frac{1}{\sqrt{\overline{d}_i + 1}}.$$

The further m by m blocks correspond to the edges of the complementary graph $\overline{G}$: let the block corresponding to the edge $\{i,j\}$ be $\alpha_i C_j$ in A_i, $\alpha_j C_i$ in A_j, and the zero matrix otherwise. Then, $(A_1,\dots,A_n)$ is a matrix orthonormal representation of G, so

$$\max_{1\le i\le n} \frac{1}{(A_i A_i^T)_{11}} \ge \hat{\vartheta}(G).$$

On the other hand,

$$\max_{1\le i\le n} \frac{1}{(A_i A_i^T)_{11}} = \max_{1\le i\le n} \frac{\overline{d}_i + 1}{(C_i C_i^T)_{11}} = \overline{d}_{\max} + 1$$

(note that the matrices C_i are orthogonal so the matrix $C_i C_i^T$ is the identity matrix). We obtained $\overline{d}_{\max} + 1 \ge \hat{\vartheta}(G)$. Similar construction on the dual side shows that $\sum_{i\in V(G)} 1/(d_i+1) \le \check{\vartheta}(G)$. The weak sandwich theorem now follows from Lemma 3.3 and the obvious inequalities $\check{\vartheta}(G) \ge \vartheta'(G)$, $\hat{\vartheta}(G) \le \vartheta(G)$. □

Note that instead of the matrices C_i in the above construction we can also use matrices D_i with the following properties: the matrices D_i are orthogonal; the matrices $D_i^T D_j$ are symmetric and have zero trace ($i \ne j$). (The only change is that the block corresponding to the edge $\{i,j\}$ is $\alpha_i D_j$ in A_i and $-\alpha_j D_i$ in A_j.)

It is an open problem to characterize the numbers m such that there exist m by m matrices $D_1,\dots,D_n$ with the properties described above; but any power of 2, greater than or equal to n meets the requirements: n words of the same length $\log_2 m$, and made up from the letters a and d (or a and c) translate into appropriate matrices (see the proof of Lemma 3.2).

Using simultaneous diagonalization (see [12]), the open problem described above can be cast also in the following form: characterize the numbers (m,n) such that there exists a matrix $M \in \{\pm 1\}^{m\times n}$ such that $M^T M = mI$. This is a subproblem of the Hadamard's Maximum Determinant Problem (see [5]); the Hadamard conjecture in an equivalent form states that the (m,n) pairs satisfying the requirements are:

- $(m,1)$ such that $m \ge 1$;
- $(m,2)$ such that $m \ge 2$ is even;
- (m,n) such that $m \ge n$ and $m \equiv 0 \pmod 4$.

4 New proof for the Caro-Wei theorem

In this section we will prove the Caro-Wei theorem (Theorem 1.2), the counterpart of Brooks' theorem (Theorem 1.1). We also describe the counterpart of Turán's theorem.

First we will show that

$$\alpha(G) \ge \sum_{i=1}^{n} \frac{1}{d_i + 1} \tag{3}$$

holds. We apply induction on the cardinality n of $V(G)$. In the case when $n = 1$, the statement is trivial; in what follows we will suppose that the number of vertices is $n > 1$ and that for graphs with smaller number of vertices the inequality (3) already holds. Note that we can suppose also that the graph G is α-critical (that is leaving out any edge the stability number becomes larger). In fact, otherwise delete edges from the graph while this operation does not change the stability number. In the end we get an α-critical graph, and the value on the right hand side of (3) became larger, while the value on the left hand side of (3) stayed the same. We can suppose also that the graph G is connected: if it has more than one components, then by induction the inequality (3) holds true for its components, and this implies the validity of (3) for the whole graph. Hence, it is enough to consider the case when the graph G is α-critical and connected.

Let v be a vertex of G such that $d_v = d_{\max}$. It is easy to prove that there exists a stable set of the size $\alpha(G)$ such that it does not contain the vertex v (see Exercise 8.12 in [9]). In fact, otherwise every maximum stable set in G contains the vertex v. As G is α-critical, so leaving out an edge $\{v, v'\}$, in the resulting graph there exists a stable set S' of the size $\alpha(G) + 1$ containing both v and v'. Then, $S' \setminus \{v\}$ is a maximum stable set in G; contradicting the indirect assumption.

Let us denote by $G - v$ the graph with vertex-set $\{1, \ldots, n\} \setminus \{v\}$, and with edge-set $\{\{i, j\} \in E(G) : i, j \neq v\}$. Then $\alpha(G - v) = \alpha(G)$. By induction, for the graph $G - v$ (3) holds, that is

$$\alpha(G - v) \geq \sum_{i \in N(v)} \frac{1}{d_i} + \sum_{i \notin N(v),\, i \neq v} \frac{1}{d_i + 1}. \tag{4}$$

As $d_v \geq d_i$ for all $i \in V(G)$, we have

$$\frac{1}{d_i} \geq \frac{1}{d_i + 1} + \frac{1}{d_v(d_v + 1)} \quad (i \in N(v)) \tag{5}$$

Writing this bound into (4) we obtain the following inequality:

$$\alpha(G - v) \geq \sum_{i \in N(v)} \frac{1}{d_i + 1} + \sum_{i \notin N(v),\, i \neq v} \frac{1}{d_i + 1} + \frac{1}{d_v + 1}.$$

As $\alpha(G - v) = \alpha(G)$, this inequality is in fact (3), and the first half of Theorem 1.2 is proved.

To prove the second half of the theorem we will show that if

$$\alpha(G) = \sum_{i=1}^{n} \frac{1}{d_i + 1} \tag{6}$$

holds then the graph G is the disjoint union of cliques (the other direction is obvious). Again we apply induction on n. Note that if (6) holds then the graph G is α-critical (otherwise G would have an edge such that after deleting this edge the stability number stays unchanged, while the value on the right hand side of (6)

becomes larger, contradicting (3)). We can suppose also that G is connected (if (6) holds then it holds for the components also). Thus it suffices to prove that if the graph G is α-critical and connected, furthermore (6) holds then G is a clique.

Let v be the same point as in the first half of the proof, and again consider the graph $G - v$. Let us denote by $\delta(G)$ the sum on the right hand side of (6). As we have seen it in the first half of the proof,

$$\alpha(G) = \alpha(G - v) \geq \delta(G - v) \geq \delta(G).$$

As now $\alpha(G) = \delta(G)$, we have equalities instead of inequalities, that is

$$\alpha(G) = \alpha(G - v) = \delta(G - v) = \delta(G).$$

It follows from the $\delta(G - v) = \delta(G)$ equality that $d_i = d_v$ $(i \in N(v))$ (as otherwise (5) would hold with strict inequality). Moreover by the $\alpha(G-v) = \delta(G-v)$ equality and by induction the graph $G - v$ is the disjoint union of cliques. As the graph G is connected, the set $N(v)$ intersects with all of these cliques. Let us choose one of the cliques, and a vertex $i \in N(v)$ from this clique. Then d_i equals d_v as well as the cardinality of the clique. Hence the components of $G - v$ all have the same cardinality d_v. Then $\alpha(G - v) = (n - 1)/d_v$. If the graph $G - v$ would have more than one components then we could chose from each component a vertex from $\overline{N}(v)$. These vertices together with the vertex v would constitute a stable set in G with cardinality larger than $\alpha(G - v)$. This would contradict the fact that $\alpha(G - v) = \alpha(G)$, so $G - v$ is a clique with cardinality d_v with vertices in $N(v)$. Thus the graph G is a clique, and the proof of the second half of Theorem 1.2 is finished also. □

Note that the bound in Brooks' theorem, $\chi(G) \leq d_{\max} + 1$, is obvious (it can be proved using a simple greedy coloring algorithm), and together with (3) and the sandwich theorem gives

$$\sum_{i=1}^{n} \frac{1}{d_i + 1} \leq \alpha(G) \leq \vartheta(G), \vartheta'(G) \leq \chi(\overline{G}) \leq \overline{d}_{\max} + 1;$$

we obtained the third proof of the weak sandwich theorem.

We remark that Turán's theorem can be derived as a consequence of the Caro-Wei theorem, see [1]. Here the graph $T_{n,m}$ is defined as follows: Divide the vertex set $V(T_{n,m}) := \{1, \ldots, n\}$ into m disjoint subsets $\bar{S}_1, \ldots, \bar{S}_m$ such that the cardinality of $\bar{S}_i$ and $\bar{S}_j$ differ by at most one for each $i \neq j$. Then the edge set of the graph $T_{n,m}$ is

$$E(T_{n,m}) := \cup_{\ell=1}^{m} \{\{i, j\} \subseteq \bar{S}_\ell : i \neq j\}.$$

Corollary 4.1. *(Turán) Let G be a simple graph on n vertices with stability number $\alpha(G) \leq m$. Minimizing the number of the edges of G under these assumptions, the unique extremal graph is $T_{n,m}$.*

The following corollary describes the counterpart of Turán's theorem which in turn is a simple consequence of Brooks' theorem.

Corollary 4.2. *Let G be a simple graph on n vertices with chromatic number $\chi(G) \geq m$. Then the number of the edges of G is at least $m(m-1)/2$. Equality holds if and only if G is the disjoint union of a clique and a stable set on m and $n-m$ vertices, respectively.*

Proof. It is well-known that the number of the edges of any simple graph G on n points is at least $\chi(G)(\chi(G)-1)/2$, so it is enough to prove that if the number of the edges is $m(m-1)/2$ and the chromatic number is m then G is isomorphic with the graph described in the statement.

Hence, we can suppose that the vertex set is the disjoint union of m stable sets with exactly one edge going between each two of them. Let us choose a connected component G' of the graph G such that its chromatic number is $\chi(G') = m$. By Brooks' theorem, then

$$m = \chi(G') \leq d_{\max}(G') + 1 \leq d_{\max}(G) + 1 \leq m,$$

and we can see that G' is a clique on m vertices; the statement is proved. □

Finally, we mention an open problem. Wilf proved the following result (see [2]): the chromatic number $\chi(G)$ is at most $\alpha_{\max}+1$ (where $\alpha_{\max}$ denotes the maximum eigenvalue of the adjacency matrix of G), with equality for a connected graph G if and only if the graph is a clique or an odd cycle. As $\alpha_{\max} \leq d_{\max}$ always holds (with equality for a connected graph if and only if the graph is regular), Wilf's theorem is stronger than Brooks' theorem. It would be interesting to see how Theorem 1.2 could be strengthened using spectral information. (The bound $n/(\alpha_{\max}+1)$ [18] is not a strengthening of the Caro-Wei bound, as — using the convexity of the function

$$x \mapsto \frac{1}{d_G^T x + 1} \ (0 \leq x \in \mathcal{R}^n),\ d_G = (d_1, \ldots, d_n)^T,$$

and Rayleigh's theorem [14] — it can be easily shown that

$$\frac{n}{\alpha_{\max}+1} \leq \sum_{i=1}^{n} \frac{1}{d_i+1}$$

holds, with equality if and only if the graph is regular.)

5 Conclusion

In this paper we presented a new proof for the counterpart of Brooks' theorem (the Caro-Wei theorem) concerning a simple lower bound on the stability number. As a consequence of the sandwich theorem, Brooks' theorem, and the Caro-Wei theorem we derived a weaker version of the sandwich theorem. For this weak sandwich theorem we gave another two, more direct proofs also, which are based on linear algebraic lemmas and new descriptions of the Lovász number.

Acknowledgements. I thank the two anonymous referees for their remarks that helped me to improve the presentation of the paper.

References

[1] Alon, N. and Spencer, J. H. *The Probabilistic Method.* John Wiley & Sons, New York, 1992.

[2] Cvetković, D. M., Doob, M., and Sachs, H. *Spectra of Graphs.* Academic Press, New York, 1979.

[3] Edwards, C. S. and Elphick, C. H. Lower bounds for the clique and the chromatic numbers of a graph. *Discrete Applied Mathematics*, 5:51–64, 1983.

[4] Geramita, A. V. and Pullman, N. J. A theorem of Hurwitz and Radon and orthogonal projective modules. *Proceedings of the American Mathematical Society*, 42(1):51–56, 1974.

[5] Jablonszkij, SZ. V. and Lupanov, O. B., editors. *Diszkrét Matematika a Számítástudományban.* Műszaki Könyvkiadó, Budapest, 1980.

[6] James, I. M. *The Topology of Stiefel Manifolds.* Cambridge University Press, Cambridge, 1976.

[7] Knuth, D. The sandwich theorem. *Electronic Journal of Combinatorics*, 1:1–48, 1994.

[8] Lovász, L. On the Shannon capacity of a graph. *IEEE Transactions on Information Theory*, IT-25:1–7, 1979.

[9] Lovász, L. *Combinatorial Problems and Exercises.* Akadémiai Kiadó, Budapest, 1979.

[10] Lovász, L. Semidefinite programs and combinatorial optimization. In Reed, B. A. and Sales, C. L., editors, *Recent Advances in Algorithms and Combinatorics*, CMS Books in Mathematics, Springer, pages 137–194, 2003.

[11] Meurdesoif, P. Strengthening the Lovász $\theta(\overline{G})$ bound for graph coloring. *Mathematical Programming A*, 102:577–588, 2005.

[12] Praszolov, V. V. *Lineáris Algebra.* Typotex Kiadó, Budapest, 2005.

[13] Radon, J. Lineare scharen orthogonaler matrizen. *Abhandlungen aus dem Matematischen Seminar der Hamburgischen Universität*, 1:1–14, 1923.

[14] Strang, G. *Linear Algebra and its Applications.* Academic Press, New York, 1980.

[15] Ujvári, M. *A Szemidefinit Programozás Alkalmazásai a Kombinatorikus Optimalizálásban.* ELTE Eötvös Kiadó, Budapest, 2001.

[16] Ujvári, M. A note on the graph-bisection problem. *Pure Mathematics and Applications*, 12(1):119–130, 2002.

[17] Ujvári, M. On a closedness theorem. *Pure Mathematics and Applications*, 15(4):469–486, 2006.

[18] Wilf, H. S. Spectral bounds for the clique and independence numbers of graphs. *Journal of Combinatorial Theory B*, 40:113–117, 1986.

Received 29th November 2011

Acta Cybernetica **20** (2012) 515–537.

Joint Perception in Agent Communication*

László Z. Varga†

Abstract

Correctness of agent communication requires that the communicating agents share a common ontology. Most of the ontology merging approaches assume that there is a global, "god's eye" view which is a combination of the concepts in the ontology of the individual agents. These approaches admit that the agents may have different views and try to resolve the differences within the limits of the global view which contains only the concepts based on the individual perceptions of the agents. In this paper we introduce the notion of *joint perception* in order to enrich the available concepts in the global view and we introduce the notion of *conceptualization based on joint perception* in order to enable the agents to resolve the differences of their views by introducing new concepts. We propose an *incremental ontology negotiation protocol for the conceptualization based on joint perception* and demonstrate it in a blocks world. With this work we develop new insights into ontology merging and negotiation for agent communication by defining a formal realization proposal for emergent semantics.

Keywords: distributed artificial intelligence, correctness of agent communication, ontology merging and negotiation, perception and conceptualization, joint perception, conceptualization based on joint perception, ontology negotiation protocol

1 Introduction

Agents can communicate with each other only if they have a common language. This means in computing terms that agents must merge their ontology into a common ontology. Ever since more than one computer system existed, ontology merging has been a fundamental issue. In the beginning the problem to be solved was the migration of data from one system to another, then the interoperability of computer systems was in focus and recently researchers started to investigate automated ontology negotiation. Most of the approaches assume that there is a global view which contains the concepts of the systems under investigation and the goal of ontology

*This work was supported by TÁMOP-4.2.2/B-10/1-2010-0030 and TÁMOP-4.2.1./B-09/1/KMR-2010-0003. This work is part of COST Action IC0801: Agreement Technologies. Many thanks to the anonymous reviewers for their valuable comments that improved the paper.

†Faculty of Informatics, Eötvös Loránd University, E-mail: lzvarga@inf.elte.hu

merging and negotiation is to discover and learn the concepts not present in one agent but present in the other. If the concepts in the ontologies of the agents are not completely compatible, then the merging methods try to resolve the contradiction to achieve a consistent global view. The approaches assume that if the ontologies of the agents are merged in this way and the agents perceive the concepts correctly, then the agents are able to communicate and work together using the merged ontology. The perception of the agents is a critical point in the above reasoning and has not been studied in the same detail as the other points of ontology merging and negotiation. In this paper we are going to investigate how perception of agents influences the agents' ability to merge and negotiate their ontologies. We propose an ontology negotiation protocol to discover new concepts with the help of joint perception in ontology merging and negotiation. The proposed ontology negotiation protocol may help agent communication, however the main goal of the paper is to better understand the role of perception in ontology merging and negotiation.

1.1 Semantics in Agent Communication

In order to be able to discuss the role of perception in agent communication, we are going to use the knowledge formalization approach by Genesereth and Nilsson in [6]. Formalization of knowledge consists of a conceptualization and an ontology. The instances of the real world are first conceptualized and then formally encoded in an ontology used by the agents in their communication as shown in Figure 1.

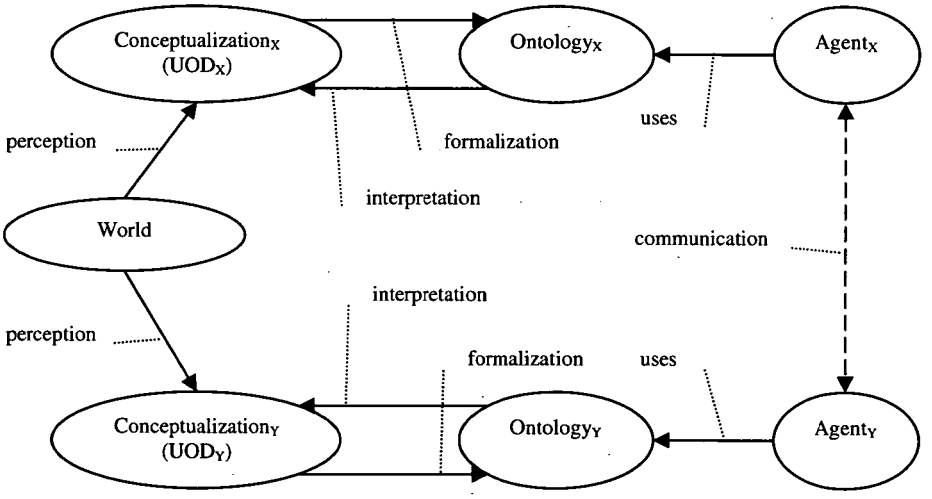

Figure 1: Semantic meaning in agent communication

When agents communicate with each other, they want to send to each other statements about the real world which is shown on the left hand side of Figure 1. The relevant instances perceived by each agent are conceptualized in the corresponding conceptualizations: Conceptualization$_X$ for Agent$_X$ and Conceptu-

alization$_Y$ for Agent$_Y$. In accordance with Genesereth and Nilsson [6], the conceptualization consists of the universe of discourse (UOD), the functional basis set and the relational basis set. The conceptualization is informal and its elements are not formally named. There can be different conceptualizations for the same world. An example of Genesereth and Nilsson [6] for agent specific conceptualization is the wave and particle conceptualization of light, where the different conceptualizations explain different aspects of the behavior of light. In that example we can observe that a conceptualization depends not only on the interest of the agent, but also on the perception capability of the agent.

In addition to the dependence on perception capability, conceptualization is context dependent as well. Context dependence can be observed in different domains and it is expressively described in the domain of image databases by Santini et al. [10]:

> "The full meaning of an image depends not only on the image data, but on a complex of cultural and social conventions in use at the time and location of the query, as well as on other contigiencies of the context in which the user interacts with the database. This leads us to reject the somewhat Aristotelean view that the meaning of an image is an immanent property of the image data. Rather, the meaning arises from a process of interpretation and is the result of the interaction between the image data and the interpreter."

While the conceptualization differences due to agent interest differences are usually mentioned in research papers that deal with ontology merging and negotiation, the conceptualization differences due to context dependence and the limitation of the perception capabilities of the agents are not in focus. We are going to focus on this perception dependence aspect of conceptualization in this paper.

Once the agent has its conceptualization, the conceptualization is formalized in an ontology that names the objects, functions and relations of the world as perceived by the agent. The ontology is represented in a formal language. The interpretation of an ontology is a mapping between the elements of the language and the elements of the conceptualization. An ontology can be mapped to the same conceptualization in several ways and an ontology can be mapped to different conceptualizations, therefore an ontology may have several interpretations. The intended interpretation is the one that the ontology developer had in mind when he/she created the ontology. Functions and relations of an ontology are satisfied by an interpretation if they are true in the corresponding conceptualization. Although several conceptualizations may satisfy an ontology, the conceptualization designated by the intended interpretation is meant to be the conceptualization to be used when agents communicate.

Communicating agents are shown on the right hand side of Figure 1. When Agent$_X$ sends a message to Agent$_Y$, then it formalizes its message using Ontology$_X$, and the message sent from Agent$_X$ to Agent$_Y$ uses the concept names used in Ontology$_X$. Agent$_Y$ decodes the message using Ontology$_Y$. This decoding can be completed if Agent$_Y$ can find the corresponding names in Ontology$_Y$. Successful

decoding of the message does not necessarily mean that the communication is semantically correct.

The communication is semantically correct only if the elements of Figure 1 match correctly, which means that if a concept c in the real world is conceptualized in Conceptualization$_X$ and formalized as c_{FX} in Ontology$_X$, then there is in Ontology$_Y$ a c_{FY} that can be mapped to c_{FX} and the intended interpretation of c_{FY} in Conceptualization$_Y$ is the conceptualization of the same c real world concept.

1.2 Ensuring Correct Semantics in Agent Communication

In order to ensure semantically correct agent communication, both agents must have a concept of the real world concept that they want to talk about. Basically this is the goal of all the ontology matching and negotiation research. The individual research reports usually focus on some of the elements of Figure 1 and assume that the others are correct.

The most studied part of Figure 1 is the right hand side, where the focus is on the formally represented ontologies. Ontology$_X$ and Ontology$_Y$ may be different, because there may be different names for the same concept in the two ontologies, or there may be different concepts in the ontologies. Semantic correctness needs that the ontologies are matched or aligned. Rahm and Bernstein [9] present a taxonomy that explains the common features of the different ontology[1] matching techniques developed in the context of ontology translation and integration, knowledge representation, machine learning, and information retrieval. An important feature of these ontology matching approaches is that they investigate the formally represented ontology on schema and instance level and try to find similarities in the formal representations based on such properties as name, description, data type, relationship types, constraints, and structure. Although the investigation of the similarities of the formal representations may indicate similarities in the concepts, the semantic correctness is not guaranteed and needs to be verified by humans. The problem with human verification is that a human is just another agent with his/her own conceptualization based on his/her own perception and we cannot be sure that this third conceptualization correctly covers and integrates the conceptualizations of Agent$_X$ and Agent$_Y$.[2]

Uschold and Gruninger [11] and Gruninger [7] propose the idea of ontological stance as a standard for semantic correctness. They go deeper than the formal representation of the ontologies and say that two ontologies are equivalent if their intended models are equivalent. Proving the equivalence of intended models is done by proving that the logical theories captured in each ontology are logically equiva-

[1] Rahm and Berstein write about schema matching, but their work can be applied to ontology matching as well.

[2] This must be one of the reasons why any attempt to create a "global ontology" of the world have failed. The ontology of Agent$_X$ and Agent$_Y$ were also created by humans, so when a third human verifies the merging of Ontology$_X$ and Ontology$_Y$, the verification is basically the same problem as merging the ontologies.

lent. Logical equivalence is verified if all statements and inferences that hold for one agent, also hold when translated into the other ontology. If the inference does not hold in the translated ontology, then the intended models are not equivalent. Unfortunately there is no procedure for generating and verifying all possible inferences for any given pair of ontologies, therefore we cannot prove semantic correctness of two ontologies, we can only prove the incorrectness if we find a conflicting inference.

The recent ontology negotiation approach includes all the elements of Figure 1. Truszkowski and Bailin [2] initiated the term ontology negotiation and recently Diggelen et al. [5] proposed an implementation of ontology negotiation for agent communication. As Williams [12] writes, a basic assumption is that both agents are able to point at instances in the real world and this can be known for both of them. Agent_X points at an instance in the real world and sends the formal representation of this instance in Ontology_X to Agent_Y. Agent_Y can see the instance in the real world and find its formal representation in Onology_Y, and thus can create a mapping between Ontology_X and Ontology_Y. The mapping method can be supported by a learning method as Williams [12] proposes or explanation based as Diggelen et al. [5] propose. Because agents point at instances in the real world, the ontology negotiation approach includes the perception part of Figure 1 as well. A basic assumption of ontology negotiation is that the agents do not have any errors in their perception of the world although their perceptions may differ. This is necessary for a successful ontology negotiation, but is it enough? Can we be sure that agents can negotiate successfully if they do not have error in their perception? In the next sections of this paper we will investigate this as well.

In the case of the above three ontology merging and negotiation approaches, the differences in the ontologies are due to the different categorizations by the agents and the goal is to match these categorizations. If the perceptions of the agents may have errors, then the agents have to eliminate the conflicting facts as well. In the example of Cholvy [3] one witness saw a dark blue car on the crime scene, while the other saw a dark green car and there were two men in it. In this case a unique consistent view of the world can be achieved by dropping some of the perceptions and keeping other perceptions. The selection is often helped by preference relations like in the work of Amgoud and Kaci [1]. In the rest of the paper we will assume that the perceptions of the agents are correct according to their conceptualization of the world.

2 The Perception Problem in a Blocks World

The above overview of ontology merging and negotiation indicates that merged ontologies and error free perceptions are needed for correct agent communication. Now we are going to investigate this in a blocks world example. Although the blocks world example below in this section has some kind of image processing flavor, we are not focusing on image processing. We are using images only because they are expressive. At the end of section 2 we will show that the blocks world example has similarities with other domains as well.

2.1 Perception Capabilities in a Blocks World

When we assume that the perception of the agent is error free, then we assume that there is an unambiguous mapping from the real world to the perception of the agent. This means that if the real world instance is in the sensing range of the agent, then the agent perceives the instance and the same real world instance always maps to the same perception of the agent.

Definition 1. *Error Free Perception: the perception of the agent is error free, if the agent perceives every real world instance that gets in the sensing range of the agent, and the same real world instance always maps to the same perception of the agent.*

We are going to investigate agents in a blocks world example, where the perception of the agents is through an image caption sensor. The sensor is able to make camera images of the real world. Figure 2 shows the perception of the real world by Agent_X. The axes x and z are not part of the perception, they are on the figure just to show the orientation. The conceptualization of this world by Agent_X consists of three blocks, no functions and two relations (square and circle) corresponding to the shape of the blocks. The formalization of this conceptualization is Ontology_X and in accordance with the formalization approach by Genesereth and Nilsson [6] it is the following[3]:

$$< \{a, b, c\}, \{\}, \{square_X, circle_X\} > \tag{1}$$

Agent_X has the following representation of the current state of the world:

$$square_X(a), circle_X(b), square_X(c) \tag{2}$$

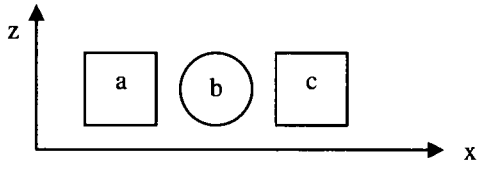

Figure 2: Perception of the blocks world scene by Agent_X

Figure 3 shows the perception of the same scene of the real world by Agent_Y. Agent_Y has exactly the same type of sensors and sensing capabilities as Agent_X, but Agent_Y has a different view. The axes y and z are not part of the perception, they are on the figure just to show the orientation. The conceptualization of Agent_Y is the same as that of Agent_X and its formalization is the same (except that the symbols are differentiated with the y index):

$$< \{a, b, c\}, \{\}, \{square_Y, circle_Y\} > \tag{3}$$

[3]We call this formal representation as Ontology_X, although it is not a complex sophisticated ontology.

However Agent_Y has the following representation of the current state of the world:

$$square_Y(a), square_Y(b), circle_Y(c) \tag{4}$$

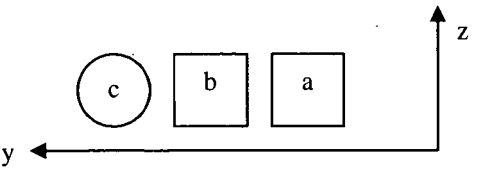

Figure 3: Perception of the blocks world scene by Agent_Y

We assume that both agents are able to point at the blocks in the real world and this can be perceived by both of them. Technically this could be implemented for example by sending a radio signal to the selected block and if the selected block receives the signal, then it emits light which is perceivable by both agents. So if Agent_X wants to point at block a, then it sends a signal to block a and Agent_Y perceives that block a is glowing. In this blocks world example the letters denote the same blocks, i.e. block a perceived by Agent_X on Figure 2 is the same as block a perceived by Agent_Y on Figure 3, block b perceived by Agent_X is the same as block b perceived by Agent_Y, and block c perceived by Agent_X is the same as block c perceived by Agent_Y.

2.2 Ontology Merging and Agent Communication in the Blocks World

We are now investigating how the different ontology merging techniques cope with the above blocks world example. In section 1.2 we have seen that there are three major approaches: the merging technique based on the formally represented ontologies (schema level matching), the verification method based on the logical equivalence of the logical theories captured in each ontology, and the ontology negotiation.

Claim 1. *Schema level matching and error free perception are not enough for conflict free agent communication.*

Proof. The merging technique based on the formally represented ontologies and systems (see Rahm and Bernstein [9]) can be on the schema or on the instance level.

The schema level matching in our blocks world example would result in stating that the ontologies of Agent_X and Agent_Y match each other, because their formal representation (1) and (3) have the same structure. After investigating the technical capabilities of the sensors of the agents and their processing software, the ontology merger would say that $square_X$ maps to $square_Y$ and $circle_X$ maps to $circle_Y$, because the agents have exactly the same type of sensors. Although both agents have error free perception and their ontologies are matched, the agents

would have problems in their communication, because if Agent_X sends the message $\textit{circle}_X\textit{(b)}$ to Agent_Y, then it would be mapped to $\textit{circle}_Y\textit{(b)}$ and it would conflict with the $\textit{square}_Y\textit{(b)}$ information of Agent_Y. So this example shows that schema level matching and error free perception are not enough for conflict free agent communication.

If schema merging is combined with instance level merging, then the above conflict on the shape of block b can be found already at the ontology merging phase. The instance level merger would not be able to find out how to map $\textit{square}_X$ to the concepts of Agent_Y, because in the case of block a the symbol $\textit{square}_X$ maps to $\textit{square}_Y$, but in the case of block c the symbol $\textit{square}_X$ maps to $\textit{circle}_Y$. The same way, the instance level merger would not be able to find out how to map $\textit{square}_Y$ to the concepts of Agent_X, because in the case of block a the symbol $\textit{square}_Y$ maps to $\textit{square}_X$, but in the case of block b the symbol $\textit{square}_Y$ maps to $\textit{circle}_X$. Therefore instance level ontology merging would fail. □

Claim 2. *The verification method based on the logical equivalence of the logical theories captured in each ontology and error free perception are not enough for conflict free agent communication.*

Proof. The ontology merging verification approach of Gruninger [7] is based on logical theories captured in the ontologies. In the case of the above blocks world example there is no complex theory captured in the formal representation of the conceptualizations, because there are no inference rules for the blocks. Therefore we can use basic statements about the state of the blocks world and general logic to verify the equivalence of the ontologies of Agent_X and Agent_Y. Let us investigate the following expressions:

$$\textit{square}_X(a) \wedge \textit{square}_X(c) \tag{5}$$

$$\textit{square}_Y(a) \wedge \textit{square}_Y(c) \tag{6}$$

$$\textit{circle}_Y(a) \wedge \textit{circle}_Y(c) \tag{7}$$

Expression (5) states that blocks a and c are both squares. Agents are able to point at the blocks, so they can identify blocks a and c. Statement (5) is evaluated true by Agent_X. The $\textit{square}_X$ symbol can be translated either to $\textit{square}_Y$ or $\textit{circle}_Y$. If $\textit{square}_X$ is translated to $\textit{square}_Y$, then we get expression (6) which is evaluated false by Agent_Y. If $\textit{square}_X$ is translated to $\textit{circle}_Y$, then we get expression (7) which is again evaluated false by Agent_Y. So expression (5) holds for Agent_X, but it does not hold in any translation into the formalization of Agent_Y. This means that the intended models of Agent_X and Agent_Y are not equivalent, the symbols of Agent_X cannot be mapped to Agent_Y and their ontologies cannot be merged. □

Claim 3. *Ontology negotiation and error free perception are not enough for conflict free agent communication.*

Proof. The ontology negotiation approach to ontology merging is somewhat similar to the instance level merging of the formally represented ontologies, but the merging is done at runtime by the agents instead of offline investigation of the formal representations. The agents point at an instance in the real world and send the representation of the instance to the other agent. In the case of the above blocks world example, Agent_X would not be able to negotiate how to map $square_X$ to the concepts of Agent_Y, because when it points at block a and sends the symbol $square_X$ to Agent_Y, then Agent_Y would map $square_X$ to $square_Y$, but when Agent_X points at block c and sends the symbol $square_X$ to Agent_Y, then Agent_Y would map $square_X$ to $circle_Y$. The same way, Agent_Y would not be able to negotiate how to map $square_Y$ to the concepts of Agent_X, because in the case of block a the symbol $square_Y$ would map to $square_X$, but in the case of block b the symbol $square_Y$ would map to $circle_X$. Therefore ontology negotiation would fail. □

Remark 1. In our simple blocks world example there are only few instances and we could easily find a mismatch in the instances in instance level schema matching or ontology negotiation, as well as conflicting statements in logic based verification. However in a complex application there may be too many instances and these types of mismatches may remain undiscovered until there is a conflict in the communication of the agents.

2.3 Why Ontology Merging Fails in the Blocks World

In the previous section we have two identical agents with two different views of the same blocks world, and the merging of their ontologies fails and the agents are not able to correctly communicate. The schema level ontology merging based on the formally represented ontologies and systems succeeds, but agent communication will not be correct. The other ontology merging approaches do not succeed in merging the ontologies, although the agents are identical. How can this be, and how can the two agents have so different perception of the same blocks world? The explanation is in the limited perception capabilities of the agents.

Definition 2. *Limited Perception Capability: given an agent that can perceive an application domain from different contexts and the perception of the agent is error free in each context, then an agent has limited perception capability if there is at least one application domain instance which maps to different perceptions of the agent in different contexts.*

Claim 4. *Agents with error free perception capabilities may not be able to resolve conflicting perceptions by choosing one of their already existing concepts, if the perception capabilities of the agents are limited.*

Proof. Figure 4 shows how the agents view the same blocks world. There are two cylinders and a cube in the blocks world. Agent_X perceives this blocks world's projection on the x-z plane (Figure 2) and Agent_Y perceives this blocks world's projection on the y-z plane (Figure 3). Agent_X perceives cylinder b as $circle_X(b)$

and Agent_Y perceives cylinder b as $square_Y(b)$. The concepts perceived by the agents are in line with the concepts of their perception devices, but not with the concepts of the real world as seen by the humans. The cylinder may be perceived by the perception devices either as a circle or a square depending on the position of the agents, and the agents are not aware of the three dimensional nature of the blocks. This limited perception capability is the root of the problems in the discussed ontology merging approaches and agent communication. □

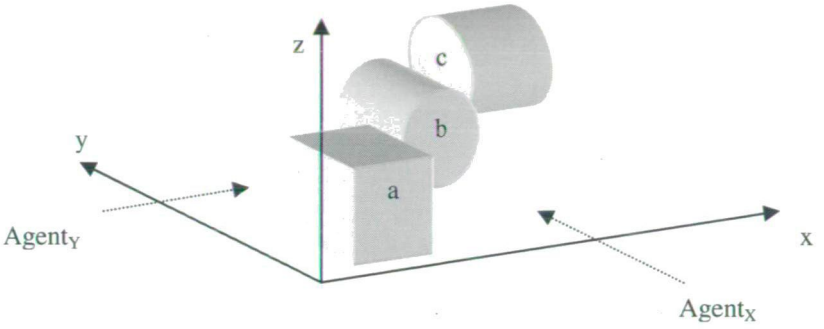

Figure 4: The blocks world scene in 3 dimensions

As long as the agents keep to the concepts of their perceptions, they will have conflicts. If they try to resolve the conflict with an argumentation framework which is based on the existing concepts of the agents, like that of Amgoud and Kaci [1], then one of the agents will be regarded more reliable than the other and the perception of the more reliable agent will win. However in this case none of the perceptions are better than the other, therefore eliminating the conflict between the agents by dropping one of the statements will not help to have a better perception of the blocks world.

2.4 The Blocks World and other Domains

We intentionally used the toy example of the blocks world in this paper, because our goal here is to have a fundamental understanding of the role of perception in ontology merging and negotiation. Once we have a clear understanding, then later ontology merging and negotiation techniques can be improved to handle more complex situations and huge amount of data.

One may think that the above blocks world example is too specific and not realistic. This is not the case, because we can easily create similar examples for the semantic concept learning application of Williams [12] in the World Wide Web domain: Agent_X is a historian expert and Agent_Y is a computer virus expert. They both have the concept of "*web page of professional interest*" and the concept of "*professionally non interesting web page*". The "*web page of professional interest*" concept corresponds to the *square* concept and the "*professionally non interesting*

web page" concept corresponds to the *circle* concept in the blocks world. A web page with title "The Trojan Horse" is similar to the *cube* in the blocks world, because both the historian and the computer expert may classify this page as "*web page of professional interest*". A web page with the title "History of Ancient Greece" is similar to the *cylinder* in the blocks world, because the historian may classify this page as "*web page of professional interest*", while the computer expert classifies this page as "*professionally non interesting web page*".

Apart from the blocks world example and its analogy above, the described phenomenon is in the heart of almost every data integration project where different representations must be merged. The views of the different developers may be different, therefore the developers of one system may identify the relevant features of a concept in a different way from the developers of the other system. This means that there may be a mismatch between the perceptions of the different developers and the concepts in the real world. For example, if there are two systems (X and Y) and the developers of both systems want to represent people and houses. The developers of system X find that the relevant features of a person are its name and social security number, while the relevant features of a house are the name of its owner, its address and the date when it was built, so they perceive the person as a *(name, number)* pair and the house as a *(name, address, date)* triple. The developers of system Y find that the relevant features of a person are its name, its address and its date of birth, while the relevant features of a house are the name of its owner and its topographical number, so they perceive the person as a *(name, address, date)* triple and the house as a *(name, number)* pair. We can see that this is similar to the blocks world example: the house corresponds to cylinder b and the person corresponds to cylinder c. The limited two dimensional perception capability is the internal representations in systems X and Y in the following way: *circle* corresponds to the *(name, address, date)* triple and *square* corresponds to the *(name, number)* pair.

Obviously the developers of system X and Y can easily understand the above toy problem and explain the differences of the concepts and their representations to each other, then add the necessary new representations and create the necessary mappings between the two systems. If there are more complex concepts and internal representations, then the developers may have difficulties in understanding and explaining the differences, therefore they need automated methods.

As we said before, the perception capability depends on the context as well, like in the case of image databases in Santini et al. [10]. If the image of a painted portrait is placed among images of other paintings (some of which are portraits and some of which are not), then an automated tool would label the images, among them the portrait, with "painting". If the image of the portrait is placed among photos and paintings of faces, then the automated tool would label the images with "face". Both perceptions are good in their context, however if we want to resolve the difference of the labellings, then the best result can be achieved if we take into account both perceptions. In the following we are going to discuss this kind of joint perception.

3 Conceptualization Based on Joint Perception

The above blocks world example clearly demonstrates that the success of ontology merging greatly depends on the perception of the agents. If the conceptualization of an agent does not describe the real world in a way that includes all the aspects necessary for the successful communication between the agents, then ontology merging fails. Although the conceptualization may be enough for a single agent to execute its own tasks, the pair of agents will not understand each other. Because the concepts in the conceptualizations of the individual agents cannot describe the real world in this case, a new conceptualization is needed. The new conceptualization may contain the concepts of the individual agents, but it should contain additional concepts as well. The new concepts are developed by combining the different views of the agents, which we call conceptualization based on joint perception.

Wooldridge [13] defines perception as the agent's capability to observe its E environment with the help of the *see* function and map it to a set of *Per* perceptions:

$$see : E \rightarrow Per \tag{8}$$

In accordance with Genesereth and Nilsson [6], the perception without formalization is the conceptualization of the agent. The formal representation of the perception follows the formalism of the ontology of the agent. Based on this, we define joint perception as two agents' capability to jointly observe their shared environment:

Definition 3. *Joint Perception: Given the E environment in which two agents* $Agent_X$ *and* $Agent_Y$ *observe the environment with the help of their* see_X *and* see_Y *functions and map the environment to two sets of perceptions* Per_X *and* Per_Y*:*

$$see_X : E \rightarrow Per_X \tag{9}$$

$$see_Y : E \rightarrow Per_Y \tag{10}$$

and the agents can communicate to each other the formalization of their perceptions with the $send_X$ *and* $send_Y$ *functions,*

then we define joint perception as the agents' capability to observe the E environment with the help of their modified $see_{X\,joint}$ *and* $see_{Y\,joint}$ *functions and map the environment to the Cartesian product of their own perception and the communicated perception of the other agent:*

$$see_{X\,joint} : E \rightarrow Per_X X send_Y(Per_Y) \tag{11}$$

$$see_{Y\,joint} : E \rightarrow Per_Y X send_X(Per_X) \tag{12}$$

The Cartesian product of the agent's own perception and the communicated perception of the other agent is called the **conceptualization based on joint perception**.

Note that the modified *see* function of the agents involves communication with the other agent, therefore the mapping result of the $see_{X\,joint}$ and $see_{Y\,joint}$ functions cannot be determined by a single agent, but by the agents together within

the framework of the **ontology negotiation protocol of the conceptualization based on joint perception** that we are going to discuss in the following sections.

With the above definition we have formally defined what Cudré-Mauroux [4] writes on emergent semantics:

> "This is a novel way of providing semantics to symbols of agents relative to the symbols of other agents with which they interact."

3.1 Ontology Negotiation for Joint Perception

Diggelen et al. [5] assume that in the ontology negotiation process there is a "god's eye view" of the conceptualizations which is the union of the individual conceptualizations of the agents. However in the above blocks world example we can enable successful agent communication only by adding new concepts to the "god's eye view": the "god's eye view" is the three dimensional view which is not perceivable by any of the agents and contains the new concept of the three dimensional cylinder.

Now we are going to extend the ontology negotiation framework of Williams [12] with a modification of the ontology negotiation protocol. We assume that both agents are able to point at instances in the real world and this can be perceived by both of them, so the agents can refer to the instances with the same instance name.

The **ontology negotiation protocol of the conceptualization based on joint perception** consists of the following steps:

1. Agent_X sends the name of one of its semantic concepts, the names of a set of instances of the semantic concept in the real world and points at the instances[4] in the real world. Agent_X repeats this message for all its semantic concepts and the corresponding sets of the instances. In the blocks world example Agent_X sends the symbol $square_X$, the names a and c, and points at blocks a and c. Then Agent_X sends the symbol $circle_X$, the name b and points at block b.

2. Agent_Y receives the semantic concept names, the instance names and observes the instances in the real world to find the corresponding semantic concept names in its internal representation. In the blocks world example Agent_Y finds that it knows that blocks a and b are $square_Y$, and block c is $circle_Y$.

3. Agent_Y builds up a joint concept name table that contains all combinations of Agent_X concept names and Agent_Y concept names, with observed instances. Agent_Y assigns new joint concept names to each row of this table. Table 1 shows this for the blocks world example. A joint concept name can be any

[4] A conceptualization consists of an universe of discourse, a functional basis set and a relational basis set. While pointing at an object is relatively easy, ponting at a functional or relational semantic concept needs further technical details of the protocol, because the agent has to point at the tuples describing the functional or relational samples. In the case of the blocks world example it is relatively easy, because we have only unary relational concepts like $square_X(a)$.

unique machine generated name, but in this blocks world example we use cube and cylinder to have correspondence with the three dimensional objects.

4. Agent$_Y$ sends the joint concept table to Agent$_X$. Agent$_X$ receives the joint concept table and incorporates the new concept names into its representation by assigning the new concept names to the real world instances. As a result, the semantic names of the concepts will be changed in the local ontology of Agent$_X$. Agent$_X$ confirms this update to Agent$_Y$.

5. Agent$_Y$ receives the confirmation and incorporates the new concept names into its local ontology, too. From this point the agents can use in their communication the new semantic names, because they are unambiguous. This means that the agents collaboratively learnt new concepts and identified the instances of the new concept based on their joint perception. These new concepts were previously unknown to them.

Table 1: Joint concept table based on joint perception.

Instances	Agent$_X$ concept	Agent$_Y$ concept	Joint concept
a	square$_X$	square$_Y$	cube
b	circle$_X$	square$_Y$	cylinder$_X$
c	square$_X$	circle$_Y$	cylinder$_Y$
—	circle$_X$	circle$_Y$	—

As Table 1 shows, the agents in the blocks world example learn the concept of the three dimensional cylinder under two new concepts names: *cylinder*$_X$ and *cylinder*$_Y$ and identified the instances of these concepts: *b* and *c* correspondingly. Although for a human observer in the three dimensional world these two types of objects are the same type of objects with different orientations, the agents assign them two different semantic names, because the joint perception of the agents is not three dimensional and the agents perceive two projections of the three dimensional space. This means that the concept names *cylinder*$_X$ and *cylinder*$_Y$ include the shape and the orientation of the three dimensional object.

The last row of Table 1 does not have any sample, therefore a semantic name is not assigned to this row. If there were a sphere in the three dimensional space, then this row would be complete.

3.2 Complexity

The conceptualization based on joint perception has the same drawback as the instance level ontology merger approaches: in a complex application there may be too many instances to check. In addition to that, the number of concepts may increase the complexity as well, so we are going to investigate this.

If the formalization of the conceptualization of Agent_X contains n semantic concept names and the formalization of the conceptualization of Agent_Y contains m semantic concept names, then the number of rows in the joint concept table will be $n * m$. In order to build up this table, Agent_X has to send n messages with n different semantic concept names to Agent_Y. Agent_Y responds to Agent_X with the joint concept table in one message containing all the maximum $n * m$ joint concept names. Altogether the messages of the proposed ontology negotiation protocol are proportional to n. Agent_Y has to find its own semantic concept name for each sample and place the sample in the corresponding row of the joint concept name table, so the computation needed to construct the joint concept name table by Agent_Y is proportional to the samples in the real world.

The ontology negotiation protocol of Williams [12] has similar complexity, because in that protocol the querying agent has to send samples for each concept name to be negotiated to the other agent, and the other agent has to decide if it can find samples for the same concept.

3.3 Ontology Negotiation as Needed

If the agents want to explore all possibilities and send to each other all concept names and their sample instances, then the joint concept table would contain all instances, as shown in Table 1. In a complex application this would be too large to send in a message, therefore we are going to modify the joint perception based ontology negotiation protocol with the lazy (or incremental) ontology alignment approach of Diggelen et. al [5]. The agents are not going to discover the whole concept space before they start communicating. Instead of that, the agents discover new concepts jointly when it is needed and they adjust their ontologies at the time when they find a mismatch in the concepts. When they discover new concepts, they incrementally solve the ontology merging problem and avoid that the reference to all instances are sent from one agent to the other.

The conceptual framework of Diggelen et. al [5] contains several ontologies for the incremental ontology alignment approach. O_X and O_Y are the local ontologies of the agents that want to align their ontologies in order to be able to communicate correctly. O_{cv} is the communication vocabulary ontology which contains the concepts that both agents understand and use for communication. O_{X-cv} is the combination[5] of O_X and O_{cv} and contains the mappings from the concepts of O_{cv} to the concepts of O_X. Similarly, O_{Y-cv} contains the mappings from the concepts of O_{cv} to the concepts of O_Y. O_{X-Y} is the combination of O_X and O_Y and contains the concepts from both agent's ontologies in a god's eye view manner. The assumption of the framework is that a) O_{X-Y} contains the union of the semantic symbols of O_X and O_Y, b) there are subset orderings of the intended interpretations of the semantic symbols in O_{X-Y}, O_X and O_Y, and finally c) the subset ordering in O_{X-Y} conforms to the subset ordering of O_X and O_Y. We will refer later to these assumptions as the "subset ordering assumption".

[5]Please note that the hyphen in O_{X-cv} denotes combination and not extraction.

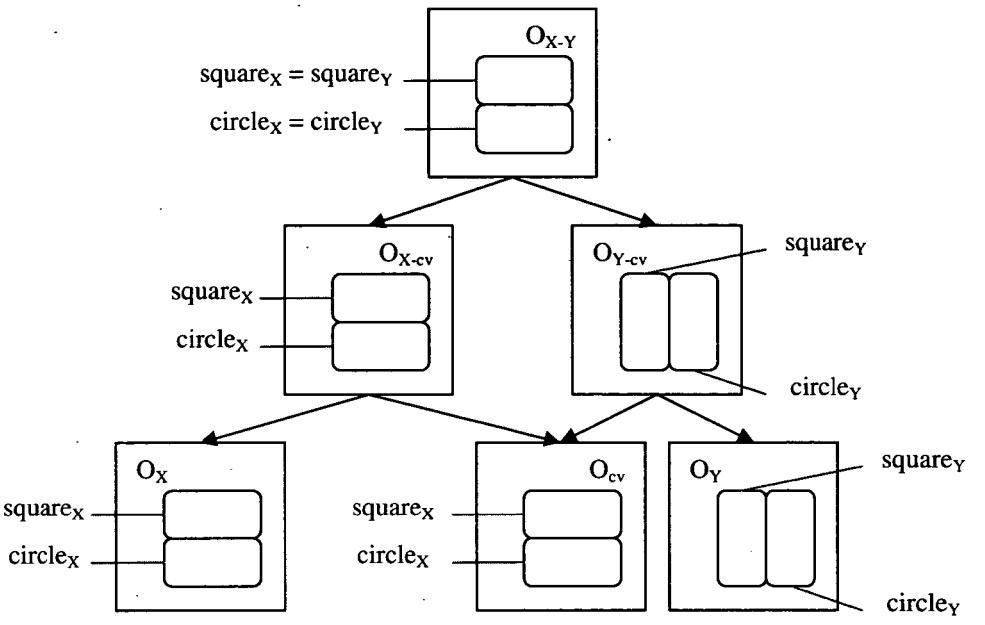

Figure 5: Ontologies according to schema level formal ontology merging in the blocks world example

Figure 5 shows these ontologies of Agent_X and Agent_Y in the blocks world example when their ontologies are merged with schema level formal ontology merging. As we said before, the schema level formal ontology merging would result in saying that the code of the agents are identical, therefore there are two concepts that are common to the agents: $square_X = square_Y$ and $circle_X = circle_Y$. This is the god's eye view and is in the O_{X-Y} ontology. The arrow from O_{X-Y} to O_{X-cv} indicates that there is a mapping from the concepts in O_{X-Y} to the concepts in O_{X-cv}: the $square_X = square_Y$ concept in O_{X-Y} is mapped to the $square_X$ concept in O_{X-cv} and the $circle_X = circle_Y$ concept in O_{X-Y} is mapped to the $circle_X$ concept in O_{X-cv}. Similarly the $square_X = square_Y$ concept in O_{X-Y} is mapped to the $square_Y$ concept in O_{Y-cv} and the $circle_X = circle_Y$ concept in O_{X-Y} is mapped to the $circle_Y$ concept in O_{Y-cv}.

The agents could use for example the symbols $square_X$ and $circle_X$ to refer to the common concepts in their communication vocabulary. This is shown in the O_{cv} ontology. The arrow from O_{X-cv} to O_{cv} indicates that there is a mapping from the concepts in O_{X-cv} to the concepts in O_{cv}: the $square_X$ concept in O_{X-cv} is mapped to the $square_X$ concept in O_{cv} and the $circle_X$ concept in O_{X-cv} is mapped to the $circle_X$ concept in O_{cv}. The arrow from O_{Y-cv} to O_{cv} indicates that there is a mapping from the concepts in O_{Y-cv} to the concepts in O_{cv}: the $square_Y$ concept in O_{Y-cv} is mapped to the $square_X$ concept in O_{cv} and the $circle_Y$ concept in O_{Y-cv} is mapped to the $circle_X$ concept in O_{cv}.

Note that the schema level formal ontology merging does not take into account the instances, therefore cannot check the assumption on the subset ordering of the intended interpretation of the semantic symbols. However if we take into account the instances and the intended interpretations as described in section 2.1, then we see that although the subset ordering assumption holds for O_X and O_Y, it does not hold for O_{X-Y}, because the sets $square_X$ and $circle_X$ are disjoint in O_X, therefore the sets $square_X = square_Y$ and $circle_X = circle_Y$ should be disjoint as well, but for example block b would be a member of both sets. This is why instance level ontology merging as well as ontology negotiations, as discussed in section 2.2, do not succeed. If we keep to the subset ordering of the original ontologies, then the agents cannot put into the merged ontology new concepts that do not conform to the original subset ordering. This means that the agents cannot discover such new concepts with the help of their joint perception capability.

Now we are going to extend the ontology negotiation protocol of the conceptualization based on joint perception (described in section 3.1) to support the incremental ontology negotiation approach of Diggelen et. al [5]. Because we want to include in the extension the possibility of learning new concepts previously unknown to the agents, we cannot keep to the subset ordering assumption and cannot directly use the ontology negotiation protocol of Diggelen et. al [5]. We will assume that the negotiation protocol of Diggelen et. al [5] will be used in the first place to determine the mapping between the O_{cv} communication vocabulary and the local ontology of the agents when there is a subset ordering of the concepts of the negotiating agents. The negotiation protocol we propose here will go to a new branch to determine a new concept when the subset ordering of the concepts of the negotiating agents does not apply or a concept mismatch is detected during communication.

Basically the incremental ontology negotiation protocol works in the following way: one of the agents proposes a concept to be added to O_{cv} and then the agents negotiate the mapping between the O_{cv} and the local ontology of the other agent. This mapping is ambiguous when the individual perceptions of the agents do not describe the real world properly and a new concept needs to be discovered based on the joint perception. Let us take the blocks world example. $Agent_X$ proposes to add the concept $square_X$ to O_{cv}. As long as $Agent_X$ points at only block a type of samples, $Agent_Y$ will map the concept $square_X$ to $square_Y$, because the perception of block a type of samples by $Agent_Y$ is $square_Y$. The result will be $square_X = square_Y$ like in the case of schema level formal ontology merging on Figure 5. However if $Agent_X$ starts to teach its $square_X$ concept with block c type of samples only, then $Agent_Y$ will map the concept $square_X$ to $circle_Y$, because the perception of block c type of objects by $Agent_Y$ is $circle_Y$. Both mappings may be sufficient for the communication of the agents as long as no instances of the $square_X$, $square_Y$ and $circle_Y$ concepts other than those used for the creation of the mapping appear in their communication. If another type of instance appears in the communication, then the new concept learning based on joint perception comes in.

The **incremental ontology negotiation protocol of the conceptualiza-**

tion based on joint perception consists of the following steps:

1. Agent$_X$ proposes to add concept name c_i to O_{cv}. If Agent$_Y$ is able to map concept name c_i into O_{Y-cv}, then the agents continue the communication (in step 3) or add other concepts to O_{cv} (this step 1 is repeated).

2. If Agent$_Y$ is not able to map concept name c_i to O_{cv}, then the agents start a new concept discovery based on joint perception (in step 4, where c_i will be denoted by c_x).

3. The agents continuously communicate with each other. If the concepts in O_{cv} correctly describe the real world for the communication, then there is no problem and normal communication goes on (this step 3 is repeated). If the agents want to extend O_{cv}, then they go to step 1 again. If the concepts in O_{cv} do not describe correctly the real world for the communication, then at some time one of the agents, let's say Agent$_X$, sends a message to the other agent, in this case to Agent$_Y$, and the message refers to a real world instance o_x of a concept c_x, the concept name c_x is in O_{cv} and mapped to c_y in O_{Y-cv}, however Agent$_Y$ discovers that according to its own perception o_x is not in concept c_y, rather in concept c_{y2}. In this case the agents start a new concept discovery based on joint perception (in step 4).

4. (The concept name c_x now denotes the conflicting concept: if we arrived here from step 2, then c_x denotes c_i of step 2, if we arrived here from step 3, then c_x denotes c_x of step 3.) Agent$_Y$ sends a message to Agent$_X$ and asks Agent$_X$ to show instances of concept c_x.

5. Agent$_X$ sends the names of a set of instances of the semantic concept c_x in the real world and points at the instances in the real world.

6. Agent$_Y$ receives the instance names and observes the instances in the real world to find the corresponding semantic concept names in its internal representation.

7. Agent$_Y$ builds up a joint concept name table that contains all combinations of c_x and Agent$_Y$ concept names, with instance names from Agent$_X$. If a new row is added to the joint concept name table, then Agent$_Y$ assigns new joint concept names to each new row of this table. A joint concept name can be any unique machine generated name.

8. The joint concept name table is permanently kept by each agent and updated each time a new concept discovery is completed. Each time the new concept discovery protocol is executed, only the newly added or modified rows are communicated by the agents in order to keep this table synchronized.

9. Agent$_Y$ sends the newly added rows of the joint concept table to Agent$_X$. Agent$_X$ receives the new rows of the joint concept table and incorporates the new concept names into O_{X-cv}. As a result some of the instances will have new semantic name. Agent$_X$ confirms this update to Agent$_Y$.

10. Agent$_Y$ receives the confirmation and incorporates the new concept names into O_{Y-cv}, too. This means that the agents collaboratively learnt new concepts based on their joint perception and at the same time jointly identified instances of the new concept as well, therefore the agents can refer to these instances in their future communication using the new concept name. These new concepts and the categorisation of the instances to these concepts were previously unknown to them. The agents add the new concepts to O_{cv} and at the same time delete c_x from O_{cv}, because c_x is replaced by the new ones. From this point the agents can continue the communication using the new semantic names (step 3) and identifiy instances of the new semantic concepts using the joint concept name table.

As an example, let's see how the above incremental ontology negotiation protocol of the conceptualization based on joint perception works in the blocks world of section 2. A sample scenario is the following:

1. Agent$_X$ proposes to add concept name $square_x$ to O_{cv} and points at block a. Agent$_Y$ maps $square_x$ to $square_y$ in O_{Y-cv}.

2. The agents start to communicate with each other. At some time Agent$_X$, sends a message to Agent$_Y$, and the message refers to block c of the concept $square_x$. The concept name $square_x$ is in O_{cv} and mapped to $square_y$ in O_{Y-cv}, however Agent$_Y$ discovers that according to its own perception, block c is in concept $circle_y$.

3. Agent$_Y$ sends a message to Agent$_X$ and asks Agent$_X$ to show samples of concept $square_x$.

4. Agent$_X$ sends the names of block a and c in the semantic concept $square_x$ and points at the sample instances in the real world.

5. Agent$_Y$ receives the instance names and observes the instances in the real world to find the corresponding semantic concept names in its internal representation.

6. Agent$_Y$ builds up a joint concept name table that contains all combinations of $square_X$ and Agent$_Y$ concept names, with sample instance names from Agent$_X$. Agent$_Y$ assigns new joint concept names to each row of this table as shown in Table 2 below. Note that Table 2 contains the categorization of the blocks a and c as well.

7. Agent$_Y$ sends the newly added rows of the joint concept table (in this case the whole table is new) to Agent$_X$. Agent$_X$ receives the new rows of the joint concept table, stores the rows of the joint concept table in its own copy of the joint concept table and incorporates the new concept names into O_{X-cv}. Agent$_X$ confirms this update to Agent$_Y$.

8. Agent$_Y$ receives the confirmation and incorporates the new concept names into O_{Y-cv}, too. This means that the agents collaboratively learnt the new concepts *cube* and *cylinder*$_Y$ together with their instances based on their joint perception and the ontologies are updated as shown in Figure 6. From this point the agents can continue the communication using the new semantic names and identify the instances of the new semantic concepts using the joint concept table.

Table 2: Joint concept table based on incremental joint perception.

Instances	Agent$_X$ concept	Agent$_Y$ concept	Joint concept
a	square$_X$	square$_Y$	cube
c	square$_X$	circle$_Y$	cylinder$_Y$

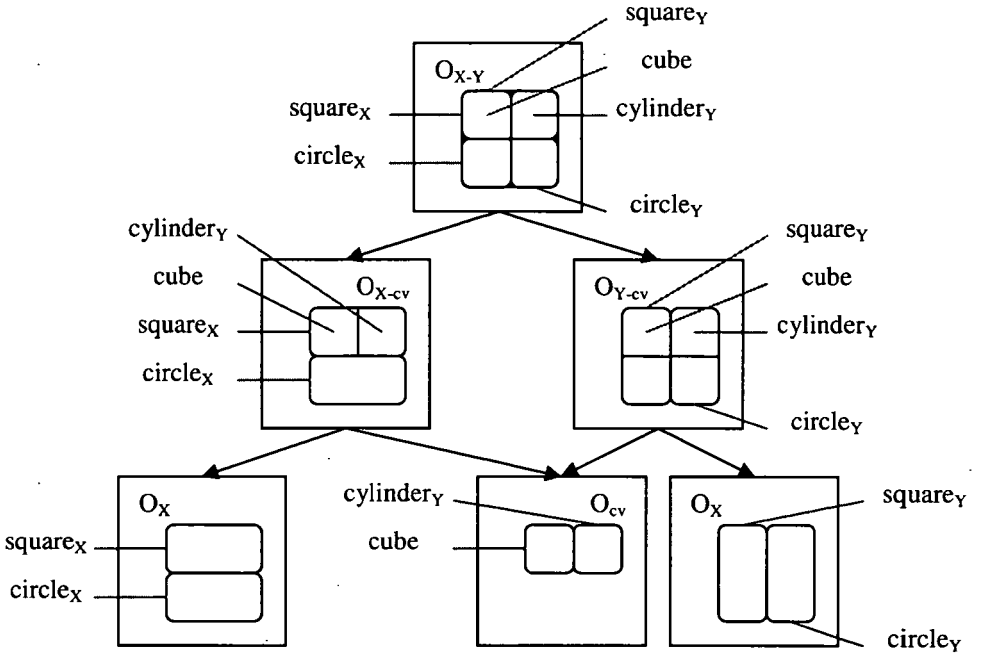

Figure 6: Ontologies of the agents after an incremental joint perception discovery cycle in the blocks world

In accordance with Table 2, in Figure 6 the O_{cv} communication vocabulary ontology contains the newly discovered concepts *cube* and *cylinder*$_Y$. Both *cube* and *cylinder*$_Y$ are included in the *square*$_X$ concept in O_{X-cv}. In O_{Y-cv} *cube* is included in *square*$_Y$ and *cylinder*$_Y$ is included in *circle*$_Y$. O_{X-Y} is the merged ontology

of the two agents, therefore it contains $square_X$ (horizontal rounded rectangle in the figure), $circle_X$ (horizontal rounded rectangle in the figure), $square_Y$ (vertical rounded rectangle in the figure), $circle_Y$ (vertical rounded rectangle in the figure), as well as the new concepts: *cube* as the intersection of $square_X$ and $square_Y$, $cylinder_Y$ as the intersection of $square_X$ and $circle_Y$.

4 Conclusions

Two agents in a multi-agent environment can communicate correctly if they share a common ontology. We can create this common ontology from the concepts perceived by the agents only if the individual perceptions of the agents correctly describe the world from both agents' view. There are two reasons why we cannot expect that the perceptions of the agents are perfect. One reason is that agents have limited perception capabilities which may be enough to perform their own tasks, but may not be correct from the point of view of the other agent. The other reason (e.g. Santini et al. [10]) is that perception is not an abstract and objective action independent from the observer, because perception depends on the complete context of the observation including the history before and after the observation, the environment of the observation, the observer and the interaction between the observer and the observed object.

So if perception is not an abstract action depending only on the perceived object, then we cannot expect that the individual perceptions of the agents always correctly describe the real world for both agents, therefore if we want to describe the world in a way that is correct from both agents' view, then we have to base the common conceptualization of the agents on the perception of both agents. This is why we introduced in this paper the notions of *joint perception* as well as *conceptualization based on joint perception.* We developed the *ontology negotiation protocol of the conceptualization based on joint perception* as an extension to the ontology negotiation framework of Williams [12]. In order to reduce instant resource usage of this ontology negotiation protocol, we developed the *incremental ontology negotiation protocol of the conceptualization based on joint perception* and showed how it fits in the incremental ontology negotiation approach of Diggelen et. al [5]. To our knowledge, this is the first work that actually describes how to create new concepts in ontology merging and negotiation for agent communication, therefore this is the first formal realization proposal for the viewpoints of Cudré-Mauroux et al. [4] on emergent semantics. In a similar way as the notion of joint intention of Jennings [8] helped to better understand cooperation in the multi-agent world, we hope that the notion of joint perception gives better insight into the role of perception in ontology merging and negotiation in multi-agent systems.

With the help of the ontology negotiation protocol of the conceptualization based on joint perception the agents can create concepts that are in line with the perceptions of both agents, therefore the ontologies of the agents can be merged into a common ontology that is suitable for both agents and the agents can correctly communicate with each other when they refer to the jointly identified concepts or

the instances of the new concepts. Although we get a common ontology with the proposed ontology negotiation protocol, the disadvantage of the proposed approach may be that the concepts newly discovered by the agents and merged into the common ontology may not be "real" concepts for the human observer. Basically a concept newly discovered by the agents is "something which is viewed in a way by one agent and viewed in another way by the other agent". Another disadvantage of the proposed approach may be that if we apply this conceptualization based on joint perception in a multi-agent environment, then we may get confusingly many new concepts in every possible pairs of agents. However, the agents may not be able to discover the same "real" concepts as the human observer, because the perceptions of the agents are limited and context based, and the agents are not able to perceive the real world in its reality. Further research will have to focus on the analysis of the proposed protocols in real settings and how to apply the ontology negotiation protocol of the conceptualization based on joint perception among three or more agents in order to support the communication of the agents. In this paper we assumed that the agents benevolently participate in the joint perception, however it would be interesting to consider the cases when the agents report false perceptions either intentionally or by mistake.

References

[1] Amgoud, Leila and Kaci, Souhila. An argumentation framework for merging conflicting knowledge bases. *Int. J. Approx. Reasoning*, 45:321–340, July 2007.

[2] Bailin, Sidney C. and Truszkowski, Walt. Ontology negotiation between intelligent information agents. *Knowl. Eng. Rev.*, 17:7–19, March 2002.

[3] Cholvy, L. A general framework for reasoning about contradictory information and some of its applications. In *Proceedings of the ECAI Workshop Conflicts Among Agents*, 1998.

[4] Cudré-Mauroux, Philippe. Emergent semantics. In Liu, Ling and Özsu, M. Tamer, editors, *Encyclopedia of Database Systems*, pages 982–985. Springer US, 2009.

[5] Diggelen, Jurriaan Van, Beun, Robbert-Jan, Dignum, Frank, Eijk, Rogier M. Van, and Meyer, John-Jules. Ontology negotiation: goals, requirements and implementation. *Int. J. Agent-Oriented Softw. Eng.*, 1:63–90, April 2007.

[6] Genesereth, M. R. and Nilsson, N. *Logical Foundations of Artificial Intelligence.* Morgan Kaufmann Publishers, San Mateo, CA, 1987.

[7] Grüninger, Michael. The ontological stance for a manufacturing scenario. In Kalfoglou, Yannis, editor, *Cases on Semantic Interoperability for Information Systems Integration: Practices and Applications*, pages 22–42. IGI Global, 2010.

[8] Jennings, Nicholas R. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artif. Intell.*, 75:195–240, June 1995.

[9] Rahm, Erhard and Bernstein, Philip A. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10:334–350, December 2001.

[10] Santini, Simone, Gupta, Amarnath, and Jain, Ramesh. Emergent semantics through interaction in image databases. *IEEE Trans. on Knowl. and Data Eng.*, 13:337–351, May 2001.

[11] Uschold, M. and Gruninger, M. Creating semantically integrated communities on the world wide web. In *Proceedings of the Semantic Web Workshop Co-located with WWW 2002 Honolulu*, 2002.

[12] Williams, Andrew B. Learning to share meaning in a multi-agent system. *Autonomous Agents and Multi-Agent Systems*, 8:165–193, March 2004.

[13] Wooldridge, Michael J. *An Introduction to Multiagent Systems.* John Wiley & Sons, Inc., Chichester, England, 2009.

Received 5th April 2011

CONTENTS

ISSN 0324—721 X

Felelős szerkesztő és kiadó: Csirik János
Nyomdai kivitelezés: E-press Nyomdaipari Kft.