Volume 28 Number 3 November 2004 ISSN 0350-5596

Informatica

An International Journal of Computing and Informatics

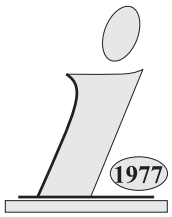
Special Issue:

Theoretical Computer Science

Guest Editors:

Boštjan Vilfan

Roberto Grossi



The Slovene Society Informatika, Ljubljana, Slovenia

EDITORIAL BOARDS, PUBLISHING COUNCIL

Informatica is a journal primarily covering the European computer science and informatics community; scientific and educational as well as technical, commercial and industrial. Its basic aim is to enhance communications between different European structures on the basis of equal rights and international refereeing. It publishes scientific papers accepted by at least two referees outside the author's country. In addition, it contains information about conferences, opinions, critical examinations of existing publications and news. Finally, major practical achievements and innovations in the computer and information industry are presented through commercial publications as well as through independent evaluations.

Editing and refereeing are distributed. Each editor from the Editorial Board can conduct the refereeing process by appointing two new referees or referees from the Board of Referees or Editorial Board. Referees should not be from the author's country. If new referees are appointed, their names will appear in the list of referees. Each paper bears the name of the editor who appointed the referees. Each editor can propose new members for the Editorial Board or referees. Editors and referees inactive for a longer period can be automatically replaced. Changes in the Editorial Board are confirmed by the Executive Editors.

The coordination necessary is made through the Executive Editors who examine the reviews, sort the accepted articles and maintain appropriate international distribution. The Executive Board is appointed by the Society Informatika. Informatica is partially supported by the Slovenian Ministry of Science and Technology.

Each author is guaranteed to receive the reviews of his article. When accepted, publication in Informatica is guaranteed in less than one year after the Executive Editors receive the corrected version of the article.

From the editors

This special issue of *Informatica* is devoted to Theoretical Computer Science, and contains a selection of papers derived from presentations at the conference *Theoretical Computer Science 2003*, which took place on October 16, 2003 within the framework of the multiconference *Information Society 2003* (October 12-17, 2003 at the Jozef Stefan Institute, Ljubljana, Slovenia).

The conference was an inaugural conference on its subject in Slovenia, and is expected to continue in the future on an annual or biannual schedule. The 2003 event attracted several interesting contributions from diverse areas of Theoretical Computer Science, and some were selected for inclusion in this issue.

The area of the *Analysis of Algorithms* is represented by the paper by Suzuki and Ibaraki, *An Average Runing Time Analysis of a Backtracking Algorithm to Calculate the Size of the Union of Cartesian Products*. The paper provides an analysis of a backtracking algorithm for the problem of calculating the size of the set $\cup_{i=1}^{n} S_{i_1} \times \ldots \times S_{i_m}$ where S_{ij} are finite sets of integers and m denotes the dimension of the space. The paper complements earlier work by the same authors, and thus provides more detailed information on the problem.

The area of *Graphs and Visualization* is represented by the papers, Bokal, Juvan, and Mohar: *A Spectral Approach to Graphical Representation of Data*, and Orbanić, Boben, Jaklič, and Pisanski: *Algorithms for Drawing Polyhedra from 3-Connected Planar Graphs*. The first considers a heuristic algorithm for the problem, given a weighted graph, find positions (x_i, y_i) of the nodes such that the relative differences between the internode distances and the edgeweights are minimised. The second represents an analysis of Tutte's method of drawing a planar graph, viewed in terms of matrix iteration and Markov chains.

Finally, the area of *Formal Languages and Compiler Construction* is represented by Mernik, Črepinšek, Kosar, Rebernak, and Žumer: *Grammar-Based Systems: Definition and Examples*, and Slivnik, and Vilfan: *Improved Error Recovery in Generated LR Parsers*. The first represents a guided tour of different applications of context-free grammars and attribute grammars. The paper does not report on novel research results; but it was felt that it does give an interesting and timely introduction to the area. The second discusses an application of a new parsing method (more precisely: an improvement of a recently reported parsing method) which generates the left parse of an input string on the basis of an LR grammar. It is described how this approach can be used to improve error reporting in parsers that are constructed with tools such as *Bison*, *Yacc*, etc.

Boštjan Vilfan, University of Ljubljana
Roberto Grossi, University of Pisa

An Average Running Time Analysis of a Backtracking Algorithm to Calculate the Size of the Union of Cartesian Products

Susumu Suzuki
Information Network Engineering, Aichi Institute of Technology,
Toyota, 470-0392 Japan
susumu-suzuki@aitech.ac.jp

Toshihide Ibaraki
Graduate School of Informatics, Kyoto University,
Kyoto, 606-8501 Japan
ibaraki@i.kyoto-u.ac.jp

Keywords: average running time, backtracking algorithm, Cartesian product

Received: February 7, 2004

Problem SUCP is the problem to calculate the size of the union of n Cartesian products, $|\cup_{i=1,\ldots,n} S_{i1}\times\cdots\times S_{im}|$, where S_{ij} are finite sets of integers and m denotes the dimension of the space. SUCP contains as a special case the problem of counting the number of unsatisfying assignments of the satisfiability problem (SAT). Therefore, SUCP is NP-hard and, in further detail, #P-complete. We presented in [7] an exact algorithm to solve SUCP, called the grouping method, and analyzed its average running time. Except for this, SUCP has been hardly studied so far. In this paper, we analyze the average running time of a backtracking algorithm to solve SUCP. For the analysis, S_{ij} are constructed by randomly selecting each element from set $D=\{1,2,\ldots,d\}$ with probability p. We show that its average running time is $O(mnd((\frac{4m(-\ln p)d}{\ln(n/(m-1))})^{m-1}\lg d+1))$ for $n\geq m$, where $\lg$ and $\ln$ denote $\log_2$ and $\log_e$ respectively. For the same instances, the average running time of the grouping method is $O(mnd(nd(1-p)+1)^{m-1})$ and that of the naive method is $O(mnd^m)$. These results indicate that the backtracking algorithm is most efficient if p and n $(\geq m)$ are large enough such that $\frac{4m(-\ln p)}{\ln(n/(m-1))}\ll\min\{1,n(1-p)\}$ holds.

Povzetek: članek opisuje časovno analizo algoritma za izračun velikosti unije kartezičnih produktov.

1 Introduction

Problem SUCP is defined as follows [7]:

SUCP Given finite sets of integers $D_j=\{1,2,\ldots,d_j\}$ and their subsets S_{ij} $(\subseteq D_j)$, where $i=1,\ldots,n$ and $j=1,\ldots,m$, calculate the size of the union of m-dimensional Cartesian products $C_i=S_{i1}\times\cdots\times S_{im}$; i.e.,

$$|C_1\cup\cdots\cup C_n| = \\ |\{(x_1,\ldots,x_m)\mid (x_1\in S_{11}\wedge\cdots\wedge x_m\in S_{1m}) \\ \vee\cdots\vee(x_1\in S_{n1}\wedge\cdots\wedge x_m\in S_{nm})\}|.$$

□

Example 1 When a problem instance is given by

$$\begin{aligned} D_1 &= D_2=\{1,2,3\},\\ S_{11} &= \{1,2\},\ S_{12}=\{1,2,3\},\\ S_{21} &= \{2,3\},\ S_{22}=\{1\},\\ S_{31} &= \{1,3\},\ S_{32}=\{1,3\}, \end{aligned}$$

the answer is

$$\begin{aligned} &|C_1\cup C_2\cup C_3|\\ &= |S_{11}\times S_{12}\cup S_{21}\times S_{22}\cup S_{31}\times S_{32}|\\ &= |\{1,2\}\times\{1,2,3\}\cup\{2,3\}\times\{1\}\\ &\quad\cup\{1,3\}\times\{1,3\}|\\ &= |\{(1,1),(1,2),(1,3),(2,1),(2,2),(2,3)\}\\ &\quad\cup\{(2,1),(3,1)\}\\ &\quad\cup\{(1,1),(1,3),(3,1),(3,3)\}|\\ &= |\{(1,1),(1,2),(1,3),(2,1),(2,2),(2,3),\\ &\quad(3,1),(3,3)\}|\\ &= 8. \end{aligned}$$

□

Applications A shop sells three kinds of T-shirts (T-shirt#1, T-shirt#2, T-shirt#3). The sizes of T-shirt#1 are small and medium, and its colors blue, green and red. Similarly, the sizes of T-shirt#2 are medium and large, and its color blue. The sizes of T-shirt#3 are small and large, and its colors blue and red. The measure of the

variety of T-shirts sold at the shop can be represented as $|\{\text{small}, \text{medium}\}\times\{\text{blue}, \text{green}, \text{red}\}\cup\{\text{medium}, \text{large}\}\times\{\text{blue}\}\cup\{\text{small}, \text{large}\}\times\{\text{blue}, \text{red}\}|$. That is, the problem of measuring the variety of T-shirts can be represented as SUCP. □

Problem SUCP contains as a special case the problem of counting the number of unsatisfying assignments of the satisfiability problem (SAT). For example, the problem of counting the number of unsatisfying assignments for DNF equation $(x_1 \wedge \overline{x_2} \wedge x_3) \vee (x_2 \wedge \overline{x_3}) \vee (\overline{x_1} \wedge x_3) =$ false is expressed as the problem SUCP for $d_1 = d_2 = d_3 = 2, C_1 = \{2\}\times\{1\}\times\{2\}$, $C_2 = \{1,2\}\times\{2\}\times\{1\}$ and $C_3 = \{1\}\times\{1,2\}\times\{2\}$, as understood by regarding "false" as 1 and "true" as 2. Therefore, SUCP is NP-hard and, in further detail, #P-complete [2].

The problem of counting the number of (un)satisfying assignments of SAT has been intensively studied, and the average running times of several algorithms (including backtracking algorithms) have been analyzed [1, 2, 3, 4, 5, 6]. On the other hand, SUCP for general $d_j(\geq 2)$ has not been studied much so far except for [7], in which we proposed an algorithm called the grouping method and analyzed an upper bound of its average running time. We are interested in algorithms that can solve SUCP with large d_j efficiently.

We explain the grouping method [7] briefly by using an example. Let C_1 and C_2 be $C_1 = S_{11}\times S_{12} = \{1,2,3\}\times\{1,2\}$ and $C_2 = S_{21}\times S_{22} = \{2,3\}\times\{1,2,3\}$. Since the decomposition of $|C_1 \cup C_2|$ over the first coordinate axis,

$$\begin{aligned} |C_1 \cup C_2| &= |\{1\}|\cdot|S_{12}| + |\{2\}|\cdot|S_{12}\cup S_{22}| \\ &\quad + |\{3\}|\cdot|S_{12}\cup S_{22}|, \end{aligned} \tag{1}$$

contains common terms $S_{12} \cup S_{22}$ in two positions in its right hand side, it can be aggregated as follows:

$$|C_1 \cup C_2| = |\{1\}|\cdot|S_{12}| + |\{2,3\}|\cdot|S_{12}\cup S_{22}|. \tag{2}$$

The grouping method calculates $|C_1 \cup C_2|$ efficiently by using this aggregated formula (2) instead of (1). To analyze the average running time, S_{ij} are constructed by randomly selecting each element from set $D_j = D = \{1, 2, \ldots, d\}$ with probability p. The average running time of the grouping method is $O(mnd(nd(1-p)+1)^{m-1})$ [7].

In addition to the grouping method, if we consider the naive method, which calculates $|C_1 \cup \cdots \cup C_n|$ by checking whether each cell $(x_1, \ldots, x_m) \in D_1 \times \cdots \times D_m$ is contained in $C_1 \cup \cdots \cup C_n$ or not, its average running time is $O(mnd^m)$.

In this paper, we propose a backtracking algorithm for the case of general $d_j \geq 2$ (denoted by SUCP-BT), which is a generalization of the basic backtracking algorithm for SAT in [6] (denoted by SAT-BT), and analyze its average running time(denoted by T), where problem instances are generated in the same manner as the above. We follow the analysis procedure in [6]. That is, we first obtain an formula (7) (using (5) and (6)) including sum ($\sum$) operations that expresses an upper bound of the average running time T, which is an generalization of the corresponding formula for SAT-BT in [6], and then transform its formula (7) into a simpler formula (23) approximately. However, the formula (7) for SUCP-BT is more complicated than the formula for SAT-BT in [6], and our transformation to obtain a simpler formula (23) is different from that in [6]. As a result of the analysis, we show that the average running time T of SUCP-BT is estimated as $O(mnd((\frac{4m(-\ln p)d}{\ln(n/(m-1))})^{m-1}\lg d + 1))$ for $n \geq m$ ((23)), where lg and ln denote $\log_2$ and $\log_e$ respectively. The result indicates that SUCP-BT is more efficient than the former algorithms (that is, the grouping method and the naive method), if p and $n(\geq m)$ are large enough such that $\frac{4m(-\ln p)}{\ln(n/(m-1))} \ll \min\{1, n(1-p)\}$ holds.

The rest of this paper is organized as follows. We present SUCP-BT in Section 2, analyze its average running time in Section 3, and give its worst-case running time in Section 4. Finally, we summarize the paper in Section 5.

2 Backtracking Algorithm

We present SUCP-BT, where $[l, r] = \{x \mid l \leq x \leq r,\ x \text{ is an integer}\}$.

step1: Put the pair (I_0, m) on top of the empty stack $STACK$, where $I_0 = [1, d_1]\times\cdots\times[1, d_m]$, and set V to 0, which is used to keep the answer $|C_1 \cup \cdots \cup C_n|$.

step2: If $STACK = \phi$, output V ($= |C_1 \cup \cdots \cup C_n|$) and halt.

step3: Take the top pair $(I, k) = ([l_1, r_1]\times\cdots\times[l_m, r_m], k)$ from $STACK$. If the m-dimensional sub-domain I is contained in at least one of Cartesian products C_i, update V by $V + |[l_1, r_1]| \times \cdots \times |[l_m, r_m]|$ and return to step2.

step4: If $|[l_j, r_j]| = 1$ for all j, return to step2. Otherwise, denote the first j in the order of $j = k+1, \ldots, m, 1, \ldots, k$, with $|[l_j, r_j]| \geq 2$ by k'. Divide the m-dimensional sub-domain I into two m-dimensional sub-domains $I' = [l_1, r_1]\times\cdots\times[l_{k'}, c-1]\times\cdots\times[l_m, r_m]$ and $I'' = [l_1, r_1]\times\cdots\times[c, r_{k'}]\times\cdots\times[l_m, r_m]$ by splitting the k'-th interval $[l_{k'}, r_{k'}]$ into two intervals $[l_{k'}, c-1]$ and $[c, r_{k'}]$, where $c = l_{k'} + \lceil (r_{k'} - l_{k'} + 1)/2 \rceil$. Put the pairs (I', k') and (I'', k') on top of $STACK$ and return to step2. □

Let us apply SUCP-BT to SUCP of Example 1 in Section 1. Figure 1 shows all of the sub-domains generated by SUCP-BT. Domain I_0 is divided into sub-domains I_1 and I_2 by a split of its first interval $[1, 3]$ (step4), since it is contained in none of Cartesian products C_1, C_2 and C_3 (step3). Sub-domain I_1 is contained in C_1. Sub-domain I_2 is divided into sub-domains I_3 and I_4 by a split of its second interval $[1, 3]$, and sub-domain I_3 divided into sub-domains I_5 and I_6 by a split of its second interval $[1, 2]$,

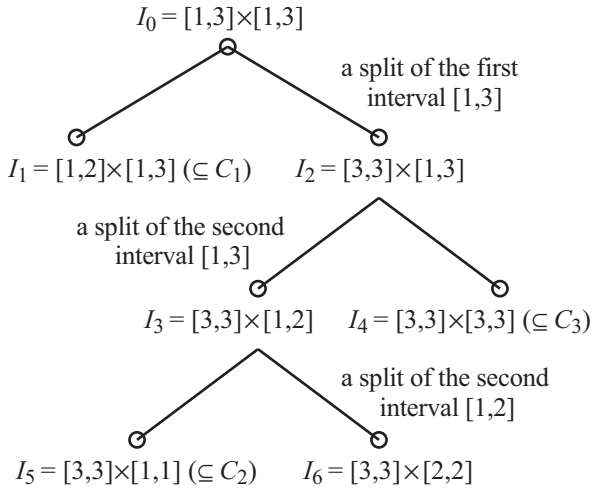


Figure 1: All of the sub-domains that the backtracking algorithm SUCP-BT generates for the problem instance of Example 1

since I_2 and I_3 are contained in none of Cartesian products. Sub-domain I_5 is contained in C_2. Sub-domain I_6 is not divided, since its size is one. Sub-domain I_4 is contained in C_3. As a result, SUCP-BT outputs

$$\begin{aligned} |C_1 \cup C_2 \cup C_3| &= |I_1| + |I_4| + |I_5| \\ &= 2 \times 3 + 1 \times 1 + 1 \times 1 \\ &= 8 \end{aligned}$$

in step2.

3 Average running time analysis

3.1 An intermediate formula for the average running time

As mentioned in Section 1, S_{ij} ($i = 1, \ldots, n, j = 1, \ldots, m$) are constructed by randomly selecting each element from set $D_j = D = \{1, 2, \ldots, d\}$ with probability p.

Let us consider the procedure obtained by modifying step3 of SUCP-BT so that the processing goes from step3 to step4 even when a sub-domain I is contained in at least one of Cartesian products (that is, the procedure obtained by modifying SUCP-BT so that a sub-domain I with $|I| \geq 2$ is always divided), and denote the tree that shows how the procedure generates sub-domains by $\mathcal{T}_1$. Nodes of $\mathcal{T}_1$ are sub-domains generated by the procedure, the root is domain $[1, d] \times \cdots \times [1, d]$, and leaves are sub-domains of which the size is one. A node I of $\mathcal{T}_1$ has its children I' and I'' when the procedure divides the sub-domain I into the sub-domains I' and I''. $\mathcal{T}_1$ has the following property:

property 1 Nodes of $\mathcal{T}_1$ that are contained in none of Cartesian products and of which the size is more than one are sub-domains that are generated and divided by SUCP-BT.

We denote the average running time of SUCP-BT by T, and also denote it by $T(d)$ when it is necessary to specify the value of d. First, we consider the case of $d = 2^h$ ($h = 1, 2, \ldots$). Then $T(2^h)$ is expressed as follows:

$$\begin{aligned} T(2^h) &= T_1(2^h)\left(1 - \left(1 - p^{2^h m}\right)^n\right) \\ &\quad + \sum_{i=0}^{h-1}\sum_{j=0}^{m-1} T_2(2^h, i, j) 2^{mi+j} \\ &\quad \times \left(1 - p^{2^{h-i-1}j + 2^{h-i}(m-j)}\right)^n. \end{aligned} \quad (3)$$

In (3), the formula $(1 - (1 - p^{2^h m})^n)$ in the first term is the probability that the domain I_0 is contained in at least one of Cartesian products, and $T_1(2^h)$ is the average of time for SUCP-BT to process I_0 in such a case (that is, the average of time for SUCP-BT to find that I_0 is contained in at least one of Cartesian products). The formula $(1 - p^{2^{h-i-1}j + 2^{h-i}(m-j)})^n$ in the second term is the probability that a sub-domain I at the depth of $mi + j$ in $\mathcal{T}_1$ is contained in none of Cartesian products, and therefore, from property 1 mentioned above, it is the probability that a sub-domain I at the depth of $mi + j$ in $\mathcal{T}_1$ is generated and divided by SUCP-BT. $T_2(2^h, i, j)$ is the average of time for SUCP-BT to process a sub-domain I in such a case. $T_2(2^h, i, j)$ includes time to find that I is contained in none of Cartesian products and time to divide I into two sub-domains I' and I''. Furthermore, if $I'(I'')$ is a sub-domain that is not divided by SUCP-BT, $T_2(2^h, i, j)$ also includes time to check whether $I'(I'')$ is contained in at least one of Cartesian products or not. $T_2(2^h, i, j)$ consists of these time.

Since $T_1(2^h)$ and $T_2(2^h, i, j)$ have the following upper bounds:

$$\begin{aligned} T_1(2^h) &= O\left(2^h mn\right), \\ T_2(2^h, i, j) &= O\left(\left(2^{h-i-1}j + 2^{h-i}(m-j)\right)n\right), \end{aligned}$$

by substituting these into (3) and replacing $1-(1-p^{2^h m})^n$ with 1, $T(2^h)$ is

$$T(2^h) = O(F(h)), \quad (4)$$

where

$$\begin{aligned} F(h) = &\\ &mn2^h + \sum_{i=0}^{h-1}\sum_{j=0}^{m-1}\left(2^{h-i-1}j + 2^{h-i}(m-j)\right) \\ &\times n2^{mi+j}\left(1 - p^{2^{h-i-1}j + 2^{h-i}(m-j)}\right)^n. \end{aligned} \quad (5)$$

Next, we consider the case of any integer d. Let $\lceil \lg d \rceil$ be denoted by a:

$$a = \lceil \lg d \rceil. \quad (6)$$

Since it holds

$$T\ (= T(d)) \quad \leq \quad T(2^a),$$

by changing h in (4) into a, the following upper bound of T is obtained:

$$T \quad = \quad O\left(F(a)\right), \tag{7}$$

where a and $F(h)$ are (6) and (5) respectively. We use the formula (7) as an intermediate formula of an upper bound of the average running time T of SUCP-BT for any integer d.

3.2 An end formula for the average running time

We transform an upper bound $O(F(a))$ ((7)) of T into a simpler formula approximately. $F(a)$ is estimated as follows:

$$\begin{aligned}
F(a) &= mn2^a \\
&\quad + \sum_{i=0}^{a-1}\sum_{j=0}^{m-1}\left(2^{a-i-1}j + 2^{a-i}(m-j)\right) \\
&\quad \times n2^{mi+j}\left(1 - p^{2^{a-i-1}j+2^{a-i}(m-j)}\right)^n \\
&\leq mn2^a \\
&\quad + \sum_{i=0}^{a-1}\sum_{j=0}^{m-1}\left(2^{a-i}j + 2^{a-i}(m-j)\right) \\
&\quad \times n2^{mi+j}\left(1 - p^{2^{a-i}j+2^{a-i}(m-j)}\right)^n \\
&= mn2^a \\
&\quad + mn\sum_{j=0}^{m-1}2^j\sum_{i=0}^{a-1}2^{a-i}2^{mi}\left(1 - p^{2^{a-i}m}\right)^n \\
&\leq mn2^a \\
&\quad + mn2^m\sum_{i=0}^{a-1}2^{a-i}2^{mi}\left(1 - p^{2^{a-i}m}\right)^n. \qquad (8)
\end{aligned}$$

By regarding 2^{a-i} in (8) as the real number $x(= 2^{a-i})$,

$$\begin{aligned}
F(a) &\leq mn2^a + mn2^m\sum_{i=0}^{a-1} x\left(\frac{2^a}{x}\right)^m (1 - p^{mx})^n \\
&\leq mn2^a + mna2^{(1+a)m} \\
&\quad \times \max\left\{\frac{(1-p^{mx})^n}{x^{m-1}} \mid 2 \leq x \leq 2^a, x \text{ is real}\right\}.
\end{aligned} \tag{9}$$

We denote the formula in the brace { } of (9) by f:

$$f(x) = \frac{(1-p^{mx})^n}{x^{m-1}}, \tag{10}$$

and evaluate the following term in (9):

$$\max\{f(x) \mid 2 \leq x \leq 2^a, x \text{ is real}\}. \tag{11}$$

We extend the domain of x from $2 \leq x \leq 2^a$ to $0 < x$. Differentiating $f(x)$ gives

$$\begin{aligned}
&f'(x) = \\
&\quad f \cdot \frac{mn(-\ln p)}{x(p^{-mx}-1)}\left\{x - \frac{(m-1)(p^{-mx}-1)}{mn(-\ln p)}\right\}.
\end{aligned} \tag{12}$$

Let the formula in the brace { } on the right hand side of (12) be denoted by $g(x)$:

$$g(x) \quad = \quad x - \frac{(m-1)(p^{-mx}-1)}{mn(-\ln p)}. \tag{13}$$

$g(x)$ and its first and second derivatives $g'(x)$ and $g''(x)$, respectively, have the following properties:

$$\begin{aligned}
g(0) &= 0, && (14)\\
\lim_{x\to\infty} g(x) &= -\infty < 0, && (15)\\
g'(x) &= 1 - \frac{m-1}{n}p^{-mx}, \\
g'(0) &= 1 - \frac{m-1}{n}, \\
\lim_{x\to\infty} g'(x) &= -\infty < 0, && (16)\\
g''(x) &= -\frac{m(m-1)(-\ln p)p^{-mx}}{n} < 0. && (17)
\end{aligned}$$

We assume that $n \geq m$, i.e., $g'(0) > 0$. From $g'(0) > 0$, (16) and (17), the equation $g'(x) = 0$ has one solution

$$\alpha \quad = \quad \frac{\ln\frac{n}{m-1}}{m(-\ln p)} \tag{18}$$

for $0 < x$, and the value of the function $g'(x)$ varies as follows: (i) $g'(x) > 0$ for $0 < x < \alpha$, $g'(x) = 0$ for $x = \alpha$, and $g'(x) < 0$ for $\alpha < x$. From property (i), (14) and (15), the equation $g(x) = 0$ has one solution β for $0 < x$, and the value of the function $g(x)$ varies as follows: $g(x) > 0$ for $0 < x < \beta$, $g(x) = 0$ for $x = \beta$, and $g(x) < 0$ for $\beta < x$. Since $f'(x)$ ((12)) and $g(x)$ ((13)) have the same sign, $f(x)$ has the maximum value for $0 < x$ when $x = \beta$:

$$\max\{f(x) \mid 0 < x,\ x \text{ is real}\} = f(\beta). \tag{19}$$

Although β cannot be represented as a brief formula, it holds

$$\alpha \quad < \quad \beta. \tag{20}$$

From (10), (18), (19) and (20), when $n \geq m$, (11) can be evaluated as follows:

$$\begin{aligned}
&\max\{f(x) \mid 2 \leq x \leq 2^a,\ x \text{ is real}\} \\
&\quad \leq f(\beta) = \frac{(1-p^{m\beta})^n}{\beta^{m-1}} < \frac{1}{\beta^{m-1}} \\
&\quad < \frac{1}{\alpha^{m-1}}.
\end{aligned} \tag{21}$$

By substituting (21) into (9),

$$\begin{aligned} F(a) &< mn2^a + mna2^{(a+1)m}\frac{1}{\alpha^{m-1}} \\ &= mn2^a + mna2^{a+1}\left(\frac{2^{a+1}}{\alpha}\right)^{m-1}. \end{aligned} \tag{22}$$

By substituting (22), (6) and (18) into $T = O(F(a))$ ((7)), an upper bound of the average running time T of SUCP-BT for $n \geq m$ is finally obtained as follows:

$$\begin{aligned} T &= O\Bigg(mn2^{\lceil \lg d\rceil} + mn\lceil \lg d\rceil 2^{\lceil \lg d\rceil+1} \\ &\quad \times \left(\frac{2^{\lceil \lg d\rceil+1}m(-\ln p)}{\ln(n/(m-1))}\right)^{m-1}\Bigg) \\ &= O\Bigg(mnd \\ &\quad \times \left(\left(\frac{4m(-\ln p)d}{\ln(n/(m-1))}\right)^{m-1}\lg d + 1\right)\Bigg). \end{aligned} \tag{23}$$

4 Worst-case running time analysis

Let the tree that shows how SUCP-BT generates sub-domains be denoted by $\mathcal{T}_2$, where an example of such a tree is Figure 1, and the depth of a sub-domain $I = [l_1, r_1] \times \cdots \times [l_m, r_m]$ in $\mathcal{T}_2$ and the sum of the sizes of all intervals $[l_j, r_j]$ of I be denoted by $\mathrm{dp}(I)$ and $\mathrm{len}(I)$ $(= \sum_{j=1}^{m} |[l_j, r_j]|)$ respectively. Since

$$\sum_{j=1}^{m} x_j \leq m\prod_{j=1}^{m} x_j$$

for any numbers $x_1, x_2, \ldots, x_m$ that are no less than 1 $(x_j \geq 1)$, it holds

$$\mathrm{len}(I) \leq m|I|. \tag{24}$$

And the sum of the sizes of all sub-domains I at the same depth in $\mathcal{T}_2$ is no more than $|I_0| = d^m$, that is, it also holds

$$\sum_{I \text{ in } \mathcal{T}_2 \text{ such that } \mathrm{dp}(I)=i} |I| \leq d^m \quad \text{for any } i. \tag{25}$$

From (24) and (25), the worst-case running time T_w of SUCP-BT is estimated as follows:

$$\begin{aligned} T_w &= O\left(\sum_{i=0}^{m\lceil \lg d\rceil} \sum_{I \text{ in } \mathcal{T}_2 \text{ such that } \mathrm{dp}(I)=i} \mathrm{len}(I)\, n\right) \\ &= O\left(\sum_{i=0}^{m\lceil \lg d\rceil} \sum_{I \text{ in } \mathcal{T}_2 \text{ such that } \mathrm{dp}(I)=i} |I|mn\right) \\ &= O\left(\sum_{i=0}^{m\lceil \lg d\rceil} mnd^m\right) \\ &= O\left(m^2nd^m \lg d\right). \end{aligned} \tag{26}$$

5 Conclusion

Problem SUCP is the problem to calculate the size of the union of n Cartesian products, $|\cup_{i=1,\ldots,n} S_{i1} \times \cdots \times S_{im}|$, where S_{ij} are finite sets of integers and m denotes the dimension of the space. We showed that the average running time of a backtracking algorithm to solve SUCP is $O(mnd((\frac{4m(-\ln p)d}{\ln(n/(m-1))})^{m-1}\lg d+1))$ for $n \geq m$, where S_{ij} are constructed by randomly selecting each element from set $D = \{1, 2, \ldots, d\}$ with probability p. The result indicates that the backtracking algorithm is more efficient than the former algorithms, i.e., the grouping method and the naive method, if p and $n(\geq m)$ are large enough such that $\frac{4m(-\ln p)}{\ln(n/(m-1))} \ll \min\{1, n(1-p)\}$ holds.

References

[1] C. A. Brown and P. W. Purdom (1981) An average time analysis of backtracking, *SIAM J. Computing*, 10(3), Society for Industrial and Applied Mathematics, pp.583–593.

[2] M. R. Garey and D. S. Johnson (1979) *Computers and Intractability – A Guide to the Theory of NP-Completeness* –, W. H. Freeman and Company.

[3] J. Gu, P. W. Purdom, J. Franco, and B. W. Wah (1997) Algorithms for the satisfiability(SAT) problem: A survey, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 35, American Mathematical Society, pp.19–151.

[4] K. Iwama (1989) CNF satisfiability test by counting and polynomial time, *SIAM J. Computing*, 18(2), Society for Industrial and Applied Mathematics, pp.385–391.

[5] P. W. Purdom (1990) A survey of average time analyses of satisfiability algorithms, *Journal of Information Processing*, 13(4), Information Processing Society of Japan, pp.449–455.

[6] P. W. Purdom and C. A. Brown (1987) Polynomial-average-time satisfiability problems, *Information Sciences*, 41, Elsevier, pp.23–42.

[7] S. Suzuki and T. Ibaraki (2003) Average running time analysis of an algorithm to calculate the size of the union of cartesian products, *Discrete Mathematics*, 273, Elsevier, pp.211–220.

A Spectral Approach to Graphical Representation of Data

Drago Bokal
Department of Mathematics,
IMFM, Jadranska 19, Ljubljana, Slovenia
drago.bokal@imfm.uni-lj.si

Martin Juvan
Department of Mathematics,
University of Ljubljana, Jadranska 19, Ljubljana, Slovenia
martin.juvan@fmf.uni-lj.si

Bojan Mohar
Department of Mathematics,
University of Ljubljana, Jadranska 19, Ljubljana, Slovenia
bojan.mohar@uni-lj.si

Keywords: Data presentation, Clustering, Graph drawing, Spectral method, Gradient method

Received: February 6, 2004

Graphical representation of relationship data is useful in several applications. Relationships among objects are modeled as a graph and the strength of relationship as weights on graph's edges. In the paper we demonstrate how the spectral method can be applied to visualize such data. Application of gradient method is suggested to fine tune the solution obtained by the spectral method.

Povzetek: članek opisuje spektralno metodo za vizualizacijo podatkov.

1 Introduction

Many applications require graphical representation of (non numerical) data. A general approach for such a task is presented. Given a data base, pairs of objects from the given data set are classified as being "close" or "far apart" by specifying a numerical value $S(x, y)$ for each such pair x, y. This value measures *similarity* of objects. This means that the value $S(x, y)$ is large for closely related objects x and y and small for very different objects. For the purpose of this paper we shall assume that S is symmetric, i.e.

$$S(x, y) = S(y, x).$$

Similarity measures can be produced in a number of ways, for example by applying the factor analysis. The goal is to represent the objects graphically (e.g. on a computer screen) in such a way that objects which are similar with respect to the similarity measure S are represented close to each other, i.e., the distance between their graphical representations is in agreement with the "similarity distance" $S(x, y)$.

Two main points of our approach rely on the spectral method where calculations are based on the eigenvalues and eigenvectors of related Laplacian matrices. General setting for this approach is surveyed by Mohar and Poljak [27]. Usually, the spectral approach can be expressed, via the Rayleigh quotient expressions for eigenvalues, as a quadratic optimization and, more generally, via semidefinite programming [1, 18, 30, 32].

Applications of eigenvalue methods in combinatorics, graph theory and in combinatorial optimization have a long history. For example, eigenvalue bounds on the chromatic number were formulated by Wilf [31] and Hoffman [23] in the 1960's. Another early application, in the area of graph partition, is due to Fiedler [15] and Donath and Hoffman [13].

An important result was the use of eigenvalues in the construction of superconcentrators and expanders by Alon and Milman [2, 3]. Isoperimetric properties of graphs and their eigenvalues play a crucial role in the design of various randomized algorithms. These applications are based on the so-called rapidly mixing Markov chains.

There is an increasing interest in the application of eigenvalues to combinatorial optimization problems. For example, Burkard, Finke, Rendl and Wolkowicz [14, 29] used an eigenvalue approach in the study of the quadratic assignment problem and general graph partition problems, Delorme and Poljak [10, 11] and Goemans and Williamson [19] in the max-cut problem, and Juvan and Mohar [24, 25] in labelling problems. Spectral partitioning, which is based on eigenvectors of Laplace eigenvalues of graphs, has proved to be a successful heuristic approach in the design of partition algorithms [7, 22, 21], in solving sparse linear systems [28], clustering [20, 6], ranking [25, 21], in graph drawing [16], automated finding of large components [12], image and video segmentation [17], etc. We refer to [27] for additional applications.

For further results, the reader may consult existing books

and survey papers, such as [8, 9, 27].

2 Problem description

Formally, the above problem can be stated as follows. Given a data set containing n objects and the similarity measure $S(x, y)$ between pairs of objects, find an embedding of the n points in the plane such that the distances between the points representing similar objects are small, while the points corresponding to objects with small similarity are far from each other. In order to formally describe the "agreement" of the similarity measure with the distances in the plane, one has to introduce a function $f : \mathbb{R} \to \mathbb{R}$ which transforms every value s of similarity into the desired distance between points representing two objects whose similarity is $S(x, y) = s$. The transformation f must satisfy the following requirements:

(F1) Monotonicity: if $s \leq s'$, then $f(s) \geq f(s')$.

(F2) Validity: $\lim_{s\to\infty} f(s) = 0$.

It can be proved easily that **(F1)** and **(F2)** together imply:

(F3) Non-negativity: $f(s) \geq 0$ for every $s \in \mathbb{R}$.

Exact choice of f depends on the data to be represented and on the properties of their similarity measure S. If similarities of distinct objects are always positive, then one may take, for example, $f(s) = 1/s$.

We consider the objects as vertices of the (complete) weighted graph G whose edge-weights are determined by S and f: the weight of the edge xy is equal to $f(S(x, y))$. This setting has an advantage that in case when similarity measure of certain pairs of objects is not defined, then the edges corresponding to such pairs can be removed from the graph.

We consider the following problem. Given is a graph $G = (V, E)$ and a weight function $w : E \to \mathbb{R}^+$ (the edge-weights). The goal is to find a mapping $\phi : V \to \mathbb{R}^2$, which assigns to every vertex of G a point in the Euclidean plane, such that the distance between the points $\phi(x)$ and $\phi(y)$ is as close as possible to the prescribed weight $w(xy)$. We refer to this problem as the *vertex placement problem*.

The vertex placement problem is an optimization problem and there are several possible choices for the energy function which is to be minimized. Our choice is described in Subsection 3.2.

3 Solving the problem

We propose the following general algorithm to find (an approximate) solution to the vertex placement problem.

Algorithm 1: Basic algorithm for solving the vertex placement problem

Input: Graph $G = (V, E)$, similarity measure $S : V \times V \to \mathbb{R}$.
Output: Placement of the vertices of G into $\mathbb{R}^2$.

Description:
Compute the edge-weights of G, $w(xy) = f(S(x, y))$, $xy \in E$.
Obtain the initial placement by the spectral method.
Run the gradient method to obtain an improved placement.
Correct the final solution.

3.1 Initial placement

To find the initial placement in Algorithm 1, the spectral method is proposed. Its formal description is given as Algorithm 2. Let us observe that this algorithm is only heuristic, and there are no theoretical guarantees that it will return a solution close to an optimum. However, as mentioned in the introduction, it behaves quite well in practice. We refer to [27] for more information.

Algorithm 2: Obtaining the initial placement of vertices

Input: Weighted graph $G = (V, E)$ with edge-weights w.
Output: Initial placement of the vertices of G in $\mathbb{R}^2$.

Description:
Compute the auxiliary matrix A from the edge-weights w by setting
$(A)_{ij} = 0$, if $i = j$ or $ij \notin E$
$(A)_{ij} = 1/w(ij)$, if $ij \in E$.
Determine the Laplace matrix L_A from A:
$(L_A)_{ii} = \sum_{k=1}^{n} (A)_{ik}$,
$(L_A)_{ij} = -(A)_{ij}$, $i \neq j$.
Compute the eigenvectors e, f of L_A corresponding to the two smallest nontrivial eigenvalues of L_A.
Set $x_i := e_i, y_i := f_i$ as the coordinates of the vertex i.

To obtain the auxiliary matrix A from the edge-weights w, one can also use the following formula:

$$\begin{aligned} (A)_{ij} &= 0, \quad \text{if } i = j \text{ or } ij \notin E \\ (A)_{ij} &= 1/(w(ij))^2, \quad \text{if } ij \in E. \end{aligned}$$

Both alternatives seem to yield good results.

If the number of objects to be represented is too large, we first apply the same spectral method to cluster the data set

into smaller cluster sets whose size fits the requirements. Particular clusters that are small enough can then be graphically represented as described above. On the other hand, the relations among clusters themselves can also be represented by the same method by defining the distance w between two clusters X, Y as

$$w(X,Y) = \frac{1}{|X||Y|} \sum_{x \in X} \sum_{y \in Y} f(S(x,y)).$$

Similar spectral approach has already been applied to clustering problem, see [4, 5].

3.2 The gradient method

The energy function we choose to minimize in solving the vertex placement problem is the sum of the squares of the relative differences between the distances implied by the current placement of the vertices, and the desired distances:

$$\mathcal{E}(\mathbf{x},\mathbf{y}) = \sum_{ij \in E} \left(\frac{w(ij) - \|(x_i,y_i) - (x_j,y_j)\|}{w(ij)} \right)^2,$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, $\mathbf{y} = (y_1, y_2, \ldots, y_n)$, (x_i, y_i) is the point in $\mathbb{R}^2$ corresponding to the vertex i, $w(ij)$ is the desired distance between vertices i and j, and E is the set of edges of the graph G on vertices $\{1, \ldots, n\}$.

The gradient method is a well-known iterative algorithm for solving optimization problems whose objective function is differentiable (see, e.g., [26]).

At each step, first the direction in which the placement will change is determined: usually, the negative gradient is taken as the direction, but in every third step the average of the last two gradients is used instead (this significantly reduces the "zig-zag" behavior which otherwise often occurs). Then the length of the step in the chosen direction is calculated. As the first approximation a step of the Newton's root finding method for the direction derivative of the energy function in the chosen direction is used (the goal is a local minimum and the derivative evaluates to zero in the minimum; the Newton's method is used to find this root). Then step of the calculated length is made in the direction of decreasing energy function. If the value of the energy function in the obtained point is higher than the current value, the length of the step is corrected: it is repeatedly multiplied by an appropriately chosen constant factor from the unit interval until the value of the energy function is lower than the current value. Additionally, if two subsequent directions differ too much (the measure is the angle between the two), the next step is to be shorter. This method is a variant of an inexact line search using the Armijo Rule as its stopping condition (cf. [26, sec. 3.2]) and combined with some heuristics.

The method stops when any of the following cases occurs:

- the norm of the direction vector is small enough,
- the number of steps exceeds maximum number allowed, or
- a certain number of the quotients of subsequent energy values are small enough.

These criteria can be tuned to achieve either higher accuracy or faster performance.

Note that the problem is invariant under translations and rotations of the plane. Thus we may fix the position of one vertex and additionally one coordinate of another vertex.

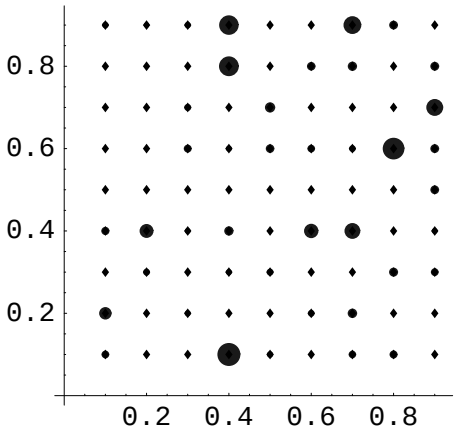


Figure 1: Performance test for random graphs with parameters $p = 0.1$, q on the x-axis and r on the y-axis ranging from 0.1 to 0.9.

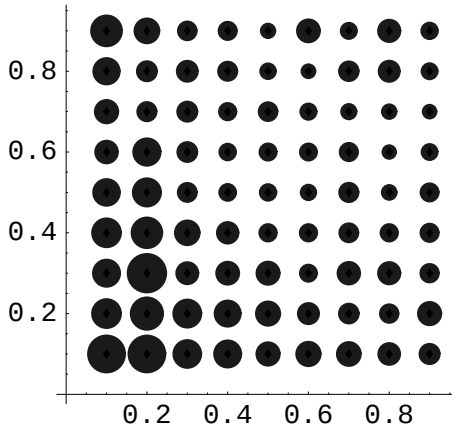


Figure 2: Performance test for random graphs ($p = 0.8$).

4 Practical considerations

The behavior of proposed algorithms was tested on several distance matrices of various sizes. For these tests we used random graphs constructed as follows: let $G = G(n,p)$ be a random graph on n vertices, where each edge is added to G with probability p. Choose randomly n points $x_1, \ldots, x_n$ in the plane, and for each edge $ij \in E_G$ let

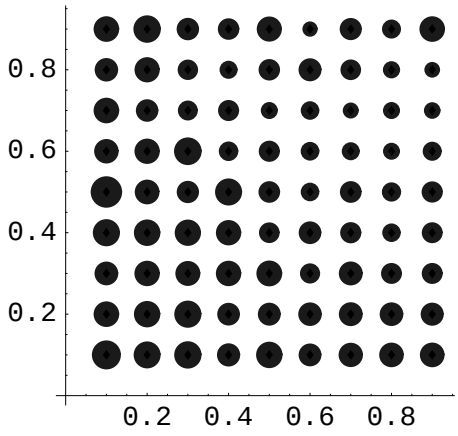


Figure 3: Performance test for random graphs ($p = 1$).

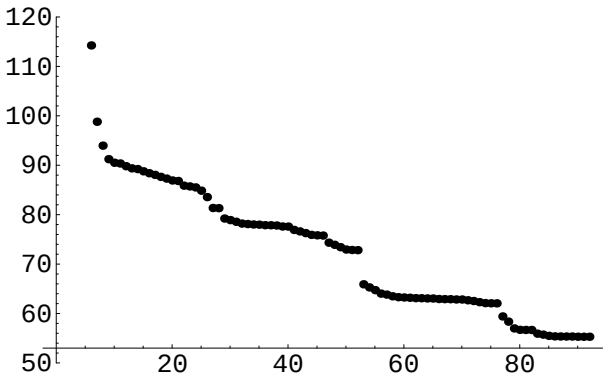


Figure 4: Decrease of the energy during iteration of the gradient method.

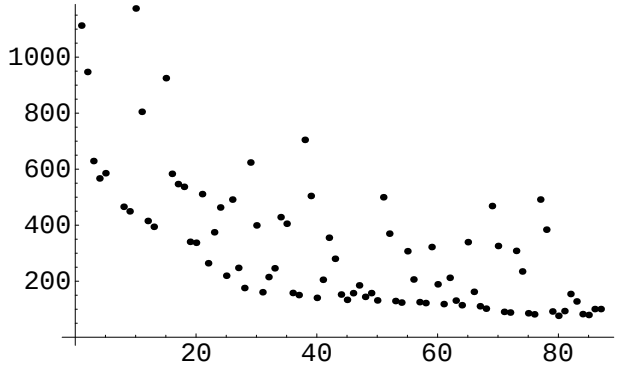


Figure 5: Changes in gradient norm during iteration of the gradient method.

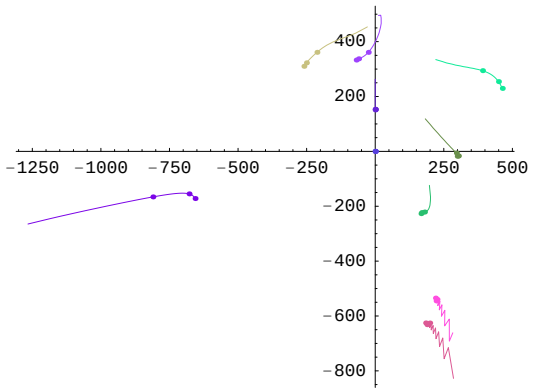


Figure 6: Traces of positions of vertices in the plane during the execution of the gradient method, initial placement using eigenvectors of smallest eigenvalues

$(D)_{ij}$ be the distance between the points x_i, x_j. With probability q the distances $(D)_{ij}$ are perturbed for a factor r (half of the distances are increased, half are decreased). Additional tests were performed using random bipartite graphs, obtained in a similar way. Such graphs appear often in the real world applications, where objects considered naturally fall into two disjoint classes.

Figures 1–3 demonstrate the results of the tests for parameters $n = 10$, q and r ranging from 0 to 0.9 by steps 0.1. The thicker the point, the more steps the gradient method required. The points were obtained averaging the results of ten independently chosen random graphs with the same parameters.

The performance analysis showed that the gradient method requires the largest number of steps when p is large with q and r being small (compare the aforementioned figures). The interpretation could be that then the optimum solution is nearly exact and hard to find, as the graph is dense. With large perturbations, the search space tends to have more local optima that are close to the initial state and the algorithm is more easily stopped in one of them. When the graph is sparse, the problem usually splits into several smaller subproblems and in general requires fewer steps of the gradient method.

Figure 4 presents the energy of the solution as a function of the number of iterations performed. It is decreasing in steps, which demonstrates that the solution may be jumping from one local optimum to another at certain points in time.

Norm of the gradient calculated using the gradient method is displayed in Figure 5. It is demonstrated that the norm of the gradient is not decreasing monotonously, however towards the local optimum its value settles and approaches 0. The large jumps in the gradient norm correspond to the bigger decreases in the value of the energy function.

The progress of the algorithm was monitored using the traces of the points in the plane. After every step of the gradient method the graph was output and its position in the plane was displayed. The positions of the same vertex in two consecutive drawings were connected by a line segment. Thus for each point we reconstructed its trace during the optimization process. Figures 6–8 display one such picture for a graph on ten vertices. These traces were displayed for various initial placements of vertices of considered graphs. Results demonstrate that the traces are shortest if we choose as the initial placement the one proposed

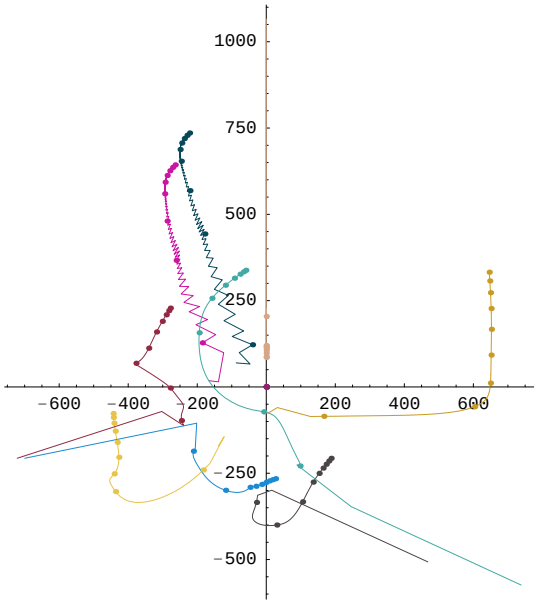


Figure 7: Traces of positions of vertices, initial placement using eigenvectors of larger eigenvalues

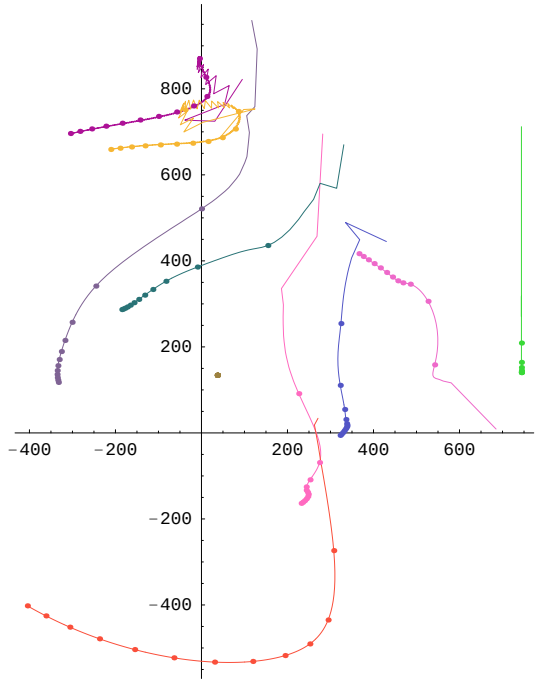


Figure 8: Traces of positions of vertices, random initial placement

by Algorithm 2 (Figure 6). This was tested both against choosing higher eigenvalues (Figure 7) and against choosing a random initial placement (Figure 8). Note that the position of the first vertex and one coordinate of the second vertex are fixed, reducing the unnecessary computational costs due to isometric transformations of the plane.

In Figure 7 certain “zig-zag" behavior of two points can be observed. It hindered the gradient method from finding the optimal solution fast. To reduce the effect of such a behavior, we slightly modified the gradient method such that in every third step, the position of vertices is updated according to the average direction of the last two gradients (see the description of the gradient method).

5 Conclusion

As demonstrated, the spectral method turns out to be well applicable to the problem of graphical representation of relations among objects. In the paper we suggested using gradient method to fine tune the solution obtained by the spectral method, however, other local optimization algorithms could be applied as well, cf. [26]. A comparison of their suitability to the described problem, as well as some rigorous analysis of convergence could be the subject of further research. One could also investigate theoretical bounds of optimality of the solutions proposed by the spectral method.

6 Acknowledgments

The authors acknowledge fruitful discussions about this subject with Robert Reinhardt. Also, programming help of Matija Mazi and Martin Milanič is greatly appreciated. We also thank the referees for their suggestions.

References

[1] F. Alizadeh, Interior point methods in semidefinite programming with applications to combinatorial optimization, SIAM J. Optimiz. 5 (1995) 13–51.

[2] N. Alon, Eigenvalues and expanders, Combinatorica 6 (1986) 83–96.

[3] N. Alon, V. D. Milman, λ_1, isoperimetric inequalities for graphs and superconcentrators, J. Combin. Theory, Ser. B 38 (1985) 73–88.

[4] M. Bolla, Spectra, Euclidean representations and clusterings of hypergraphs, Discrete Math. 117 (1993), 19–39.

[5] M. Bolla, G. Tusnády, Spectra and optimal partitions of weighted graphs, Discrete Math. 128 (1994), 1–20.

[6] P. K. Chan, M. Schlag, J. Zien, Spectral k-way ratio cut partitioning and clustering, Symp. on Integrated Systems, 1993.

[7] T.F. Chan, W.K. Szeto, On the optimality of the median cut spectral bisection graph partitioning method, SIAM J. Scientific Comput. 18 (1997) 943–948.

[8] F. R. K. Chung, Spectral graph theory, American Math. Soc., Providence, RI, 1997.

[9] D. M. Cvetković, M. Doob, H. Sachs, Spectra of graphs, Academic Press, New York, 1979; 3rd edition, Johann Ambrosius Barth Verlag, Heidelberg, 1995.

[10] C. Delorme, S. Poljak, Laplacian eigenvalues and the maximum cut problem, Math. Programming 62 (1993) 557–574.

[11] C. Delorme, S. Poljak, Combinatorial properties and the complexity of a max-cut approximation, Europ. J. Combin. 14 (1993) 313–333.

[12] C. Ding, X. He, and H. Zha, A spectral method to separate disconnected and nearly disconnected web graph components, Proc. 7th ACM Int'l Conf Knowledge Discovery and Data Mining (KDD 2001), 2001, pp. 275–280.

[13] W. E. Donath, A. J. Hoffman, Lower bounds for the partitioning of graphs, IBM J. Res. Develop. 17 (1973) 420–425.

[14] G. Finke, R. E. Burkard and F. Rendl, Quadratic assignment problem, Ann. Discrete Math. 31 (1987) 61-82.

[15] M. Fiedler, Algebraic connectivity of graphs, Czech. Math. J. 23 (98) (1973) 298–305.

[16] P. W. Fowler, T. Pisanski, J. Shawe-Taylor, Molecular graph eigenvectors for molecular coordinates, in "Graph drawing: GD'94," (R. Tamassia, ed.), Springer-Verlag, Berlin, 1995, pp. 282–285.

[17] C. Fowlkes, S. Belongie, F. Chung, and J. Malik, Spectral grouping using the Nyström method, preprint, 2002.

[18] M. Goemans, Semidefinite programming in combinatorial optimization, Math. Program. 79 (1997) 143-161.

[19] M. X. Goemans, D. P. Williamson, Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming, J. ACM 42 (1995) 1115–1145.

[20] L. Hagen, A. B. Kahng, New spectral methods for ratio cut partitioning and clustering, IEEE Trans. Computer-Aided Design 11 (1992) 1074–1085.

[21] C. Helmberg, B. Mohar, S. Poljak, F. Rendl, A spectral approach to bandwidth and separator problems in graphs, Linear and Multilinear Algebra 39 (1995) 73–90.

[22] B. Hendrickson, R. Leland, An improved spectral graph partitioning algorithm for mapping parallel computations, SIAM J. Sci. Comput. 16 (1995) 452–469.

[23] A. J. Hoffman, On eigenvalues and colorings of graphs, in "Graph Theory and Its Applications" (B. Harris, ed.), Acad. Press, 1970, pp. 79–91.

[24] M. Juvan, B. Mohar, Optimal linear labelings and eigenvalues of graphs, Discrete Appl. Math. 36 (1992) 153–168.

[25] M. Juvan, B. Mohar, Laplace eigenvalues and bandwidth-type invariants of graphs, J. Graph Theory 17 (1993) 393-407.

[26] C. T. Kelley, Iterative Methods for Optimization, SIAM, Philadelphia, 1999.

[27] B. Mohar, S. Poljak, Eigenvalues in combinatorial optimization, in "Combinatorial and Graph-Theoretical Problems in Linear Algebra," R. A. Brualdi, S. Friedland, V. Klee, Eds., IMA Volumes in Mathematics and Its Applications, Vol. 50, Springer-Verlag, 1993, pp. 107–151.

[28] A. Pothen, H. D. Simon, K.P. Liu, Partitioning sparse matrices with eigenvectors of graph, SIAM J. Matrix Anal. Appl. 11 (1990) 430–452.

[29] F. Rendl, H. Wolkowicz, Applications of parametric programming and eigenvalue maximization to the quadratic assignment problem, Math. Progr. 53 (1992) 63-78.

[30] L. Vandenberghe and S. Boyd, Semidefinite programming, SIAM Review 38 (1996) 49-95.

[31] H. S. Wilf, The eigenvalues of a graph and its chromatic number, J. London Math. Soc. 42 (1967) 330–332.

[32] H. Wolkowicz, R. Saigal and L. Vandenberghe (editors), Handbook of semidefinite programming, Theory, algorithms, and applications, International Series in Operations Research & Management Science 27, Kluwer Academic Publishers, Boston, MA, 2000.

Algorithms for Drawing Polyhedra from 3-Connected Planar Graphs

Alen Orbanić, Marko Boben, Gašper Jaklič and Tomaž Pisanski
IMFM, OTR,
Jadranska 19,
Ljubljana, Slovenia
Emails: *Alen.Orbanic@fmf.uni-lj.si*, *Marko.Boben@fmf.uni-lj.si*,
Gasper.Jaklic@fmf.uni-lj.si, *Tomaz.Pisanski@fmf.uni-lj.si*

Keywords: graph drawing, Tutte's drawing method, representation of a graph, polyhedral representations

Received: September 15, 2003

Two algorithms for producing polyhedral representations for 3-connected planar graphs are discussed in the paper. One of them uses Tutte's drawing algorithm [11] to produce a 2D drawing. Then the drawing is lifted into 3D space obtaining a polyhedral embedding. The other is a simple algorithm by G. Hart [4] for drawing canonical polyhedral representations. Some alternative aspects (physical model, Markov chain model) in algorithms for obtaining Tutte's drawings are presented and proved.

Povzetek: članek opisuje dva pristopa za grafičen prikaz planarnih grafov.

1 Introduction

A convex polyhedron can be viewed as a convex hull of its vertices and referred as $P = (\mathbf{p}_1, \ldots, \mathbf{p}_n)$, $\mathbf{p}_i \in \mathbb{R}^3$, or as an intersection of the half-spaces defined by the supporting planes of the faces using the side of $\mathbb{R}^3$ that contains the polyhedron. These are two dual definitions. For more detailed and generalized definitions of polyhedra see [12].

Any graph mentioned in the article is 3-connected planar. Vertices and edges of a polyhedron P define a skeleton graph $G(P)$ in an obvious way. The skeleton graph is obviously planar. By Balinski's theorem [12] this graph is 3-connected. In 1922 Steinitz proved that 3-connected planar graphs are exactly the skeletons of the convex 3D polyhedra. According to Whitney [2], every 3-connected planar graph has a unique embedding in the plane and the faces of the embedding are exactly the non-separating induced cycles. These faces are exactly the faces of any polyhedron with the same skeleton graph.

Given a 3-connected graph G one would like to have an algorithm to obtain a polyhedron with its skeleton G. We will call such a representation a *polyhedral representation*. A graph is given as a combinatorial structure and our algorithms return 3D coordinates of the vertices of the polyhedron. There are infinitely many polyhedral representations for a given graph. According to Koebe [5], for each polyhedron there is the canonical form, which is determined up to a rotation in 3D space. The canonical representation is especially "nice" because it possesses maximal possible geometric symmetry. Each edge of such a polyhedron touches the unit sphere in exactly one point and the center of the gravity of these touching points is the origin. If a polyhedron P is in canonical form, then the polar polyhedron is also in canonical form and the dual edges have the same touching points with the unit sphere. For a given polyhedron P containing the origin of $\mathbb{R}^3$ a polar polyhedron P^* is well defined and unique:

$$P^* = \{\mathbf{y} \in \mathbb{R}^3 \mid \langle \mathbf{x}, \mathbf{y} \rangle \leq 1 \text{ for all } \mathbf{x} \in P\}.$$

A skeleton graph of P^*, $G^* := G(P^*)$ is exactly the dual graph of the skeleton graph $G := G(P)$.

We will present some methods together with references for more detailed information. [1]

2 Tutte's algorithm

The proofs and detailed descriptions can be found in [10] and [11].

In 1963 Tutte [11] invented an interesting rubber-band method for embedding a 3-connected planar graph G into the plane. By this method one face C (on vertices $k+1, \ldots, n$, referred as the *outer cycle*) of a graph is embedded into the plane as a fixed strictly convex polygon, while the other adjacent vertices are connected with rubber-bands. On the edge ij, where i, j not both in C, there is a strictly positive stretching coefficient ω_{ij}. Later we will also assign the weights ω_{ij} to edges $ij \in C$, but we will not use them in the Tutte's method. Let $\mathbf{p}_i$ represent a position of the vertex i in the plane. The positions $\mathbf{p_i} := \mathbf{p}_i^0$, $i = k+1, \ldots, n$ are fixed. The other positions are to be calculated and lie somewhere inside the polygon. The system consisting of a rigid polygon as a frame and rubber-bands is in *equilibrium* if the following system of equations holds:

$$\sum_{ij \in E(G)} \omega_{ij}(\mathbf{p}_j - \mathbf{p}_i) = 0, \qquad i \in V(G) - V(C), \quad (1)$$

$$\mathbf{p}_i = \mathbf{p}_i^0, \quad i \in V(C).$$

[1] *Supported in part by Ministrstvo za šolstvo, znanost in šport Republike Slovenije, grant J1-6161, J2-6193.*

Tutte proved that this system has a unique solution which gives a set of positions that induces a straight line embedding of G into the plane with convex faces. Instead of solving the system of the linear equations directly, two alternative approaches can be used.

In [8] the following algorithm called *Schlegel diagram* was introduced:

ALGORITHM 1 *Fix the positions of the vertices of the outer cycle C on a convex polygon in $\mathbb{R}^2$. Assign to the other vertices random positions inside the polygon. For those vertices repeat the following steps:*

- *for each vertex i calculate the resulting force $\mathbf{F}_i$ of all adjacent rubber-bands;*
- *move each vertex i for vector $\alpha\mathbf{F}_i$, where $\alpha > 0$ is a fixed real number;*

until the displacements of all vertices are sufficently small (depends on the prescribed precision of the drawing).

We will determine the values of α which guarantee the convergence of Algorithm 1.

A *weighted Laplacian matrix* Q_ω for a graph G on n vertices and weights $\omega = (\omega_{ij})_{ij\in E(G)}$ is a $n \times n$ matrix with $(Q_\omega)_{i,j} = (Q_\omega)_{j,i} = -\omega_{ij}$ when $ij \in E(G)$, $(Q_\omega)_{i,j} = 0$, when $i \neq j$, and $ij \notin E(G)$ and $(Q_\omega)_{i,i} = -\sum_{j=1,j\neq i}^{n}(Q_\omega)_{i,j}$. Let $K = \{1,\ldots,k\}$ and $Q_{\omega,K}$ be a matrix that consists of the first k rows of Q_ω. Let

$$A = \begin{pmatrix} & -Q_{\omega,K} & \\ 0_{k,n-k} & & -I_{n-k} \end{pmatrix}, \tag{2}$$

where $0_{k,n-k}$ represents a $k \times (n-k)$ zero matrix and I_ℓ a $\ell \times \ell$ identity matrix. Let $\mathbf{x} = (x_1,\ldots,x_n)$ and $\mathbf{y} = (y_1,\ldots,y_n)$, where $\mathbf{p}_i = (x_i, y_i)$. Let $\mathbf{p}_i := \mathbf{p}_i^0$ be fixed values for $i = k+1,\ldots,n$ and $\mathbf{p}_i$ be the pairs of variables for $i = 1,\ldots,k$. Let also $\mathbf{b} = (b_1,\ldots,b_n)$, $\mathbf{c} = (c_1,\ldots,c_n)$ and $(b_i,c_i) = (0,0)$ for $i = 1,\ldots,k$ and $(b_i,c_i) = -\mathbf{p}_i^0$ for $i = k+1,\ldots,n$. The system of equations (1) can be written as:

$$A\mathbf{x} = \mathbf{b}, \qquad A\mathbf{y} = \mathbf{c}. \tag{3}$$

The iteration procedure in the Algorithm 1 can be rewritten as:

$$\begin{aligned} \mathbf{x}^{n+1} &= \mathbf{x}^n + \alpha(A\mathbf{x}^n - \mathbf{b}), \\ \mathbf{y}^{n+1} &= \mathbf{y}^n + \alpha(A\mathbf{y}^n - \mathbf{c}). \end{aligned} \tag{4}$$

Let us determine the bounds for α that ensure that the algorithm converges. It is sufficient to do this for the iteration for $\mathbf{x}$. Let us rewrite the iteration (4):

$$\mathbf{x}^{n+1} = (\alpha A + I)\mathbf{x}^n - \alpha\mathbf{b}. \tag{5}$$

Let $\rho(M)$ denote a spectral radius of a matrix M: It is well known that an iteration of this type converges iff $\rho(\alpha A + I) < 1$, see [1]. We will show that for a sufficiently small $\alpha > 0$ it follows $\rho(\alpha A + I) < 1$.

If $\sigma(A)$ is the spectrum of the matrix A, then (with slight abuse of notation) $\sigma(\alpha A + I) = 1 + \sigma(\alpha A) = 1 + \alpha\cdot\sigma(A)$. Proving that $\sigma(A)$ is strictly negative would yield the existence of the appropriate α, such that $\rho(\alpha A + I) < 1$. The matrix A is block diagonal and upper triangular with two diagonal blocks $-Q_{\omega,(K,K)}$ and $-I_k$, where $Q_{\omega,(K,K)}$ is a matrix obtained from Q_ω taking the rows and the columns in the set K in induced order. It follows that $\sigma(A) = \sigma(-Q_{\omega,(K,K)}) \cup \sigma(-I_k)$. It suffices to show that $Q_{\omega,(K,K)}$ is positive definite.

The matrix $Q_{\omega,(K,K)}$ is a principal sub-matrix of Q_ω. We will prove that $\sigma(Q_\omega) \geq 0$ and that $Q_{\omega,(K,K)}$ is non-singular. Cauchy's interlacing theorem [1] implies that $Q_{\omega,(K,K)}$ is also positive definite.

Using the Gershgorin's theorem (see [1]) it is easy to see that the eigenvalues of Laplacian matrix Q_ω are contained in $[0, 2\cdot\max_{i=1,\ldots,n}\{(Q_\omega)_{i,i}\}]$, hence $\sigma(Q_\omega) \geq 0$. By Whitney's theorem $H = G - C$ is connected. Let $Q' := Q_\omega(H)$ be a weighted Laplacian matrix of H and $Q := Q_{\omega,(K,K)}$. Then: $Q = Q' + \Delta$, where Δ is a nonnegative diagonal matrix. Since G is connected, $\Delta \neq 0$. For an arbitrary vector $\mathbf{w} \in \mathbb{R}^k$:

$$\mathbf{w}^T Q\mathbf{w} = \mathbf{w}^T Q'\mathbf{w} + \mathbf{w}^T\Delta\mathbf{w}.$$

Since eigenvalues of Q' are by Gershgorin's theorem contained in the interval $[0, 2\cdot\max_{i=1,\ldots,k}\{Q'_{i,i}\}]$, it follows that $\mathbf{w}^T Q'\mathbf{w} \geq 0$. Since H is connected, the multiplicity of the smallest eigenvalue is 1. But the smallest eigenvalue is 0 and its eigenvector is the all ones vector $\mathbf{1}$. Therefore $\mathbf{w}^T Q'\mathbf{w} = 0$ iff $\mathbf{w} = c\,\mathbf{1}$ for some $c \in \mathbb{R}$. Since Δ is a nonnegative diagonal matrix, it follows that $\mathbf{w}^T\Delta\mathbf{w} \geq 0$. But for $\mathbf{w} = \mathbf{1}$, $\mathbf{w}^T\Delta\mathbf{w} = c^2\mathbf{1}^T\Delta\mathbf{1} > 0$, if $c \neq 0$. So $\mathbf{w}^T Q\mathbf{w} > 0$ for each $\mathbf{w} \neq 0$ and Q is positive definite. The following theorem holds:

THEOREM 1 *Let G be a 3-connected planar graph. If:*

$$0 < \alpha < \frac{2}{\max\left\{1, 2\cdot\max_{i=1,\ldots,k}\left\{\sum_{ij\in E(G)}\omega_{ij}\right\}\right\}},$$

then Algorithm 1 converges to the solution of (1).

The other alternative approach comes from probability. Let us define a Markov chain $X_0, X_1, \ldots$ with transition matrix P using the graph G and the weights ω:

$$P_{ij} = \begin{cases} 0 & ij \notin E(G), \\ \frac{\omega_{ij}}{\sum_{ik\in E(G)}\omega_{ik}} & ij \in E(G) - E(C), \\ 1 & ij \in E(C). \end{cases} \tag{6}$$

After rewriting the system (1) we get:

$$\begin{aligned} \mathbf{p}_i &= \frac{\sum_{ij\in E(G)}\omega_{ij}\cdot\mathbf{p}_j}{\sum_{ij\in E(G)}\omega_{ij}}, && i \in V(G) - V(C), \\ \mathbf{p}_i &= \mathbf{p}_i^0, && i \in V(C). \end{aligned} \tag{7}$$

or

$$\begin{aligned} P\mathbf{x} &= \mathbf{x}, \\ P\mathbf{y} &= \mathbf{y}, \\ \mathbf{p}_i &= \mathbf{p}_i^0 = (x_i^0, y_i^0), \qquad i \in V(C). \end{aligned} \tag{8}$$

The vertices of the graph G are the states of the Markov chain. Let $A \subset V(G)$. The values:

$$h_i^A = \Pr(X_r \in A \text{ for some } r \geq 0 \mid X_0 = i), \tag{9}$$

are *hitting probabilities* for the set A when starting in the state i. We will write h_i^j for $h_i^{\{j\}}$. The following theorem holds (see [7]):

THEOREM 2 *A vector $\mathbf{h}^A = (h_1^A, \ldots, h_n^A)$ is a solution of a system:*

1. $h_i^A = 1$, *for* $i \in A$,
2. $\sum_{j=1}^n P_{ij} h_j^A = h_i^A$, *for* $i \notin A$,
3. $0 \leq h_i^A \leq 1$ *for all* i.

If $\widetilde{\mathbf{h}}^A$ is any other solution then $h_i^A \leq \widetilde{h}_i^A$ (inequality on components). Therefore $\mathbf{h}^A$ is a minimal solution.

Let $\mathbf{h}^j = (h_i^j)_{i=1}^n$ be the vector of hitting probabilities. By Theorem 2, $Ph^j = h^j$. Let $J = [\mathbf{h}^{k+1}, \ldots, \mathbf{h}^n]$ be a matrix with vectors $\mathbf{h}^j$ as columns. Hence $PJ = J$.

Let $\mathbf{x}^0 = (x_{k+1}^0, \ldots, x_n^0)$ and $\mathbf{y}^0 = (y_{k+1}^0, \ldots, y_n^0)$ be the coordinates of vertices on the outer polygon. Then:

$$\begin{aligned} PJ\mathbf{x}^0 &= J\mathbf{x}^0, \\ PJ\mathbf{y}^0 &= J\mathbf{y}^0, \end{aligned} \tag{10}$$

and

$$\begin{aligned} (J\mathbf{x}^0)_j &= x_j^0, \\ (J\mathbf{y}^0)_j &= y_j^0, \quad j = k+1, \ldots, n. \end{aligned} \tag{11}$$

Thus $\mathbf{p}_i = ((J\mathbf{x}^0)_i, (J\mathbf{y}^0)_i)$ is a solution of the system (7) and it is unique by Tutte.

The following theorem holds:

THEOREM 3 *Let $G = (\{1, \ldots, n\}, E(G))$ be a 3-connected planar graph, $C = (k+1, \ldots, n)$ be one of its faces, and $\mathbf{p}_i^0$, $i = k+1, \ldots, n$, be the fixed coordinates of the vertices of C forming a convex polygon in the cyclical order of C. Let P be a transition matrix for the Markov chain (X_r) and let h_i^j be the hitting probabilities for entering into vertices $j = k+1, \ldots, n$, starting from the vertex $i \in V(G)$. Then the Tutte's drawing can be obtained in the following way:*

$$\begin{aligned} \mathbf{p}_i &= \sum_{j=k+1}^{n} h_i^j \mathbf{p}_j^0, \qquad i \in V(G) - V(C), \\ \mathbf{p}_i &= \mathbf{p}_i^0, \qquad i \in V(C). \end{aligned}$$

The theorem was originally conjectured by T. W. Tucker.

To obtain the hitting probabilities h_i^j one can make sufficiently large number of the following experiments: start in i and make a series of steps until a state in C is reached. The frequency of the walks ending in j is an approximation for h_i^j.

It can be easily verified that

$$\lim_{n\to\infty} (P^n)_{ij} = h_i^j, \quad i \notin C, j \in C.$$

Hence the hitting probabilities can be obtained by calculating P^r, r large.

3 Lifting of a Tutte's Drawing

A detailed procedure with proofs can be found in [10].

A weighted embedded 3-connected planar graph G with weights ω and positions $\mathbf{p}_1, \ldots, \mathbf{p}_n$ of the vertices is in equilibrium if:

$$\sum_{ij \in E(G)} \omega_{ij}(\mathbf{p}_j - \mathbf{p}_i) = 0, \quad \text{for all } i \in V(G). \tag{12}$$

The corresponding weight ω is called an *equilibrium weight*. Maxwell-Cremona's theorem states that if for an embedded 3-connected graph with convex faces there exists an equilibrium weight with strictly negative weights on the edges of the outer face and strictly positive weights on the inner edges, then the drawing can be lifted to a polyhedron. A Tutte's drawing with the corresponding weights on the edges that are not in the outer face has all the vertices not on the outer face in an equilibrium. To use a Maxwell-Cremona's theorem there should be negative equilibrium weights on the edges of the outer face. A simplified version of the theorem will be used to solve that problem.

PROPOSITION 4 *Let G be a 3-connected planar graph embedded with one outer face f_1 on a convex polygon with all other faces ($f_2, \ldots, f_m$) as convex polygons inside the outer polygon. Let G^* be the geometric dual of G. Let $e = ij \in E(G)$ and let $e^* = fg$ be the corresponding dual edge, such that if one traverses from i to j in the embedding of G, the face f is on the left side. Let (i, j, f, g) denote an* oriented patch *and let $\mathbf{q}_{f_1} = (0, 0, 0)$. Then for $2 \leq k \leq m$ the unique assignments $\mathbf{q}_{f_k}$ exist, such that:*

$$\mathbf{q}_f - \mathbf{q}_g = \omega_{ij}(\mathbf{p}_i \times \mathbf{p}_j),$$

where ω is an equilibrium weight and $\times$ denotes a cross product.

For the proof see [10].

Using vectors $\mathbf{q}$ one can define a function on the interior of the outer polygon:

$$z(\mathbf{x}) = \langle \mathbf{x}, \mathbf{q}_{f_i} \rangle, \quad \text{for } \mathbf{x} \in f_i \subset \mathbb{R}^2. \tag{13}$$

It can be proved that the function $z(\mathbf{x})$ is linear, continuous, convex, and the image of each face lies on some plane. It

can be seen that if we have a Tutte's drawing with a triangle as the outer face, then using (12) on the Tutte's drawing, one can calculate the remaining negative weights on the edges of the triangle. The surface defined by the graph of the function $z(\mathbf{x})$ together with a patch in the place of the triangle determines a hull of the polyhedral representation of G. Using the fact that if G is 3-connected planar then either G or G^* has a triangle as a face (see [6]). This implies the following algorithm:

ALGORITHM 2 *1. Determine the faces of G (some planarity algorithm).*

2. If one of the faces is a triangle, use G otherwise use G^*.

3. Draw a Tutte's drawing.

4. From the Tutte's drawing and using Proposition 4 determine vectors $\mathbf{q}$.

5. Determine the vertices of the polyhedron using the function $z(\mathbf{x})$.

6. If G was used, we have a polyhedral embedding. Otherwise move the polyhedron so that the center of the gravity of the vertices lies in the origin. Calculate the polar polyhedron.

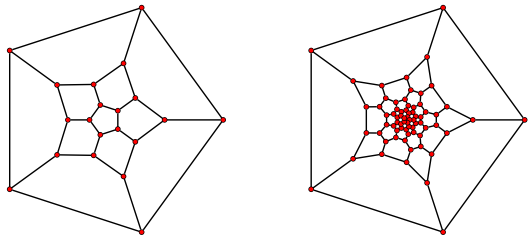


Figure 1: Dodecahedron and Fullerene C60 drawn by Tutte's method.

4 Canonical polyhedra

To produce the canonical polyhedral representation one can use methods for determining primal-dual circle packing (PDCP) of 3-connected planar graph. Having PDCP one can easily obtain the canonical polyhedral representation using the inverse of the stereographic projection on the unit sphere. One of the algorithms is due to Mohar [6] (circle packing in the plane or on the sphere). The other, which works in 3D and on more or less small graphs, is due to G. Hart [4]. We present some slight improvements for the latter algorithm. The Hart's algorithm reads:

ALGORITHM 3 *1. Start with a "good approximation" of a polyhedron that has edges as near as possible to the unit sphere. For instance, get the representation from the Algorithm 2, move the center of gravity into the origin, and project vertices from the origin to the unit sphere.*

2. Repeat the following procedure until the positions of the vertices are precise enough:

(a) for each edge e calculate the point $\mathbf{p}_e$, which is the closest to the origin. If the edge is not at the distance 1 to the origin, move the endpoints of e for a vector $\alpha(1 - \|\mathbf{p}_e\|)\mathbf{p}_e$, where $0 < \alpha < 1$.

(b) for each face calculate the approximate plane of the vertices on the face. If some vertex v is at the distance $d(v)$ from the plane, move v towards the plane in the direction of the normal for a magnitude $\beta d(v)$, $0 < \beta < 1$.

Algorithm 3 works well on small graphs. It seems to work well on cubic graphs and perhaps on graphs with lower degrees of the vertices, but the convergence might be poor, especially on the larger graphs. It behaves very poorly (does not even converge) if degrees are rather high (≥ 4). More or less we tested the performance on cubic graphs. If the cubic graph is large the convergence is very poor. To improve the convergence on cubic graphs we first use Algorithm 3 for a few iterations. Then at each iteration we calculate the vertices of the approximate polar (from approximate faces) and the dual edges. Then we check if the current dual edges are perpendicular to the original edges. If not, the vertices are moved in a manner of rotation for some small magnitude that depends on the difference between the current angle and $\pi/2$. This brings a small improvement to the convergence of the algorithm.

Both methods significantly depend on α, β and similar parameters in the improved algorithm. Unfortunately we were not able to prove the convergence for any set of the parameters.

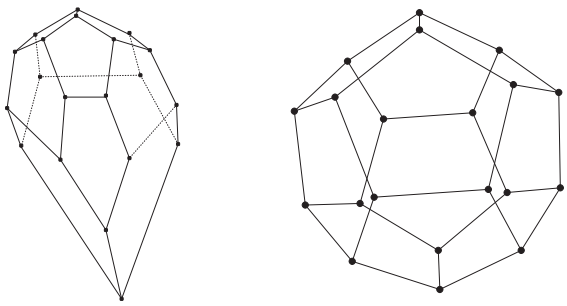


Figure 2: A dodecahedron obtained by lifting Tutte's drawing of its dual graph (left) and a dodecahedron in the canonical form (right).

References

[1] J. W. Demmel, *Applied numerical linear algebra*, SIAM Society for Industrial and Applied Mathematics, 1997.

[2] R. Diestel, *Graph Theory*, Springer, 1997.

[3] C. D. Godsil, G. F. Royle, *Algebraic Graph Theory*, Graduate Texts in Mathematics, 207, Springer-Verlag, 2001.

[4] G. W. Hart, *Calculating Canonical Polyhedra*, Mathematica in Education and Research, Vol 6 No. 3, Summer 1997, pp. 5–10.

[5] P. Koebe, *Kontaktprobleme der Konformen Abbildung*, Ber. Sächs. Akad. Wiss. Leipzig, Math.-Phys. Kl., **88** (1936), 141–164.

[6] B. Mohar, *A polynomial time circle packing algorithm*, Discrete Math. **117** (1993), 257–263.

[7] J. Norris, Markov Chains, Cambridge University Press, 1999.

[8] T. Pisanski, B. Plestenjak, A. Graovac: *NiceGraph Program and its Applications In Chemistry*, Croat. Chem. Acta **68** (1995), 283–292.

[9] T. Pisanski et al., VEGA *program*, `http://vega.ijp.si/Htmldoc/Vega03.html`

[10] J. Richter-Gebert, *Realization Spaces of Polytopes*, Lecture Notes in Mathematics, Vol. 1643, Springer-Verlag, Berlin, 1996.

[11] W.T. Tutte, *How to draw a graph*, Proc. London Math. Soc. **13** (1963), 743–767.

[12] G. M. Ziegler, *Lectures on Polytopes*, Graduate Texts in Mathematics 152, Springer-Verlag, New York 1995, Revised edition 1998.

Grammar-Based Systems: Definition and Examples

Marjan Mernik, Matej Črepinšek, Tomaž Kosar, Damijan Rebernak and Viljem Žumer
University of Maribor, Faculty of Electrical Engineering and Computer Science
Smetanova ulica 17, 2000 Maribor, Slovenia
{marjan.mernik, matej.crepinsek, tomaz.kosar, damijan.rebernak, zumer}@uni-mb.si

Keywords: context-free grammars, attribute grammars, grammar-based systems

Received: January 30, 2004

Formal language theory is an important part of theoretical computer science and has also been applied in many practical applications. The importance of context-free grammars and attribute grammars for compiler construction and automatic generation for compilers/interpreters is already well known. However, grammars can be found in many other applications which are not as clearly related to their original application - language description and implementation. We call such systems grammar-based systems. No general comparison and classification has been done until now despite these systems having existed for a long time. The aim of this paper is to introduce and popularize grammar-based systems.

Povzetek: članek opisuje definicije in primere sistemov s formalnimi slovnicami.

1 Introduction

This paper emphasizes grammars, especially context-free grammars and attribute grammars. Their importance for compiler construction is already well known. However, from formal language definitions (e.g. attribute grammars) many other language-based tools can be automatically generated [7, 9], such as: pretty printers, syntax-directed editors, type checkers, dataflow analyzers, partial evaluators, debuggers, profilers, test case generators, visualizers, animators, and documentation generators. In most of these cases, the core language definitions have to be augmented with tool-specific information. In other cases, only a part of the formal language definition is sufficient for automatic tool generation, or implicit information must be extracted from the formal language definition in order to automatically generate a tool.

Moreover, grammars can be found in many other applications which are not as clearly related to their original application - language description and implementation. We call such systems grammar-based systems (GBSs). Some papers describing particular approaches even contain this word in their titles (e.g. [4], [33], [36]). However, there is no exact definition nor comparison and classification for such systems. Since grammar-based systems are mainly unnoticed, there is also a lack of identifying benefits of such systems. The aim of this paper is to remedy this situation by defining, introducing and popularizing grammar-based systems. The benefits of GBSs are identified and clearly stated.

The organization of the paper is as follows. Section 2 presents the original application of grammars, namely automatic generation of compilers/interpreters, and other language-based tools using our compiler generator LISA [22]. In section 3, we introduce GBSs, define them and their application areas, followed by presentation of practical examples in section 4. Concluding remarks and future research work are given in section 5.

2 Original Application of Grammars

The original application of grammars is a notation for language description and its implementation [1]. What do we gain by formalizing the syntax and semantics of a programming language? The following benefits are identified:

- The language definition standardizes the language. This is important to programmers, who need to write syntactically and semantically correct programs and understand them without any doubt about their meaning. It is also important to language implementors, who need to write a correct compiler/interpreter of the specified language.

- The language definition allows a formal analysis of its properties, such as whether the definition is LL(k) grammar and L-attributed grammar. This contributes to better syntax and semantics of the programming language. The programming language that has been formally designed is more regular, has less exceptions and is easier to learn.

- The language definition enables us to systematically derive the implementation of a language, such as a LR(k) parser and attribute evaluator. Moreover, such an implementation can be automatically obtained. In this case, the language definition is used as an input to a compiler generator system. Researchers have

recognized the possibility that many other language-based tools could be generated from a formal language definition. Therefore, many tools not only automatically generate a compiler/interpreter, but also complete language-based environments [7]. Such automatically generated language-based environments include editors, type checkers, debuggers, various analyzers, and animators.

Automatic generation of compilers/interpreters and other language-based tools using our LISA compiler generator are presented in the rest of this section. To support incremental language development [21] and educational activities in teaching “Compiler construction" course [23] the LISA (*Language Implementation System based on Attribute grammars*) tool was developed [22]. LISA is a compiler-compiler, or a system that automatically generates a compiler/interpreter from attribute grammar-based language specifications. The specification of a toy language SELA (Simple Expression Language with Assignments) is given below, in order to illustrate the LISA style.

```
language SELA {
 lexicon {
   Number     [0-9]+
   Identifier [a-z]+
   Operator   \+ | :=
   ignore     [\0x09\0x0A\0x0D\ ]+
 }

 attributes Hashtable *.inEnv, *.outEnv;
            int *.val;

 rule Start {
   START ::= STMTS compute {
     STMTS.inEnv = new Hashtable();
     START.outEnv = STMTS.outEnv;
  };
 }
 rule Statements {
   STMTS ::= STMT STMTS compute {
               STMT.inEnv = STMTS[0].inEnv;
               STMTS[1].inEnv = STMT.outEnv;
               STMTS[0].outEnv = STMTS[1].outEnv;
             }
           | STMT compute {
               STMT.inEnv = STMTS[0].inEnv;
               STMTS[0].outEnv = STMT.outEnv;
             };
 }
 rule Statement {
   STMT ::= #Identifier \:= EXPR compute {
              EXPR.inEnv = STMT.inEnv;
              STMT.outEnv = put(STMT.inEnv,
                #Identifier.value(), EXPR.val);
            };
 }
 rule Expression {
   EXPR ::= EXPR + EXPR compute {
              EXPR[2].inEnv = EXPR[0].inEnv;
              EXPR[1].inEnv = EXPR[0].inEnv;
              EXPR[0].val = EXPR[1].val+
                            EXPR[2].val;
            };
 }
 rule Term1 {
   EXPR ::= #Number compute {
              EXPR.val = Integer.valueOf(
                  #Number.value()).intValue();
            };
 }
 rule Term2 {
  EXPR ::= #Identifier compute {
             EXPR.val = ((Integer)EXPR.inEnv.get(
               #Identifier.value())).intValue();
            };
 }
}
```

LISA automatically generates a SELA compiler/interpreter from this specification An example of a program written in the SELA language is shown in Figure 1.

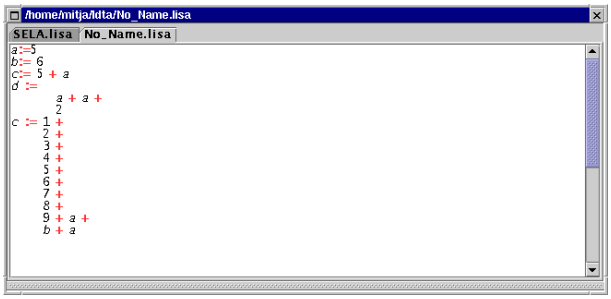


Figure 1: Language knowledgeable editor

LISA also automatically generates other tools, such as language knowledgeable editors and various inspectors (e.g. finite state automata visualizator (Figure 2), syntax and semantic tree animators (Figure 3)) that are useful for understanding the behavior of the generated language compiler/interpreter. A LISA-generated language knowledgeable editor is aware of the regular definitions of the language lexicon. Therefore, it can color the different parts of a program (comments, operators, reserved words) to enhance the understandability and readability of programs. In Figure 1 the operators in the SELA program are recognized while editing and displaying in a different color.

3 Grammar-Based Systems: Definition

As already mentioned, grammars can be found in many other systems than those described in section 2. These systems do not focus on language definition and implementation, but on solving various other problems. We call such systems grammar-based. The essential characteristics of GBSs is comprised in the following definition:

A grammar-based system is any system that uses a grammar and/or sentences produced by this grammar to solve various problems outside the domain of programming language definition and its implementation. The vital component of such a system is well structured and expressed with a grammar or with sentences produced by this grammar in an explicit or implicit manners.

The key characteristic of GBSs according to our definition is a grammar which presents a kernel for the problem solving part of the system (application). Without this grammar part, the system becomes less general, and by lacking an important generic functionality, it can become usable

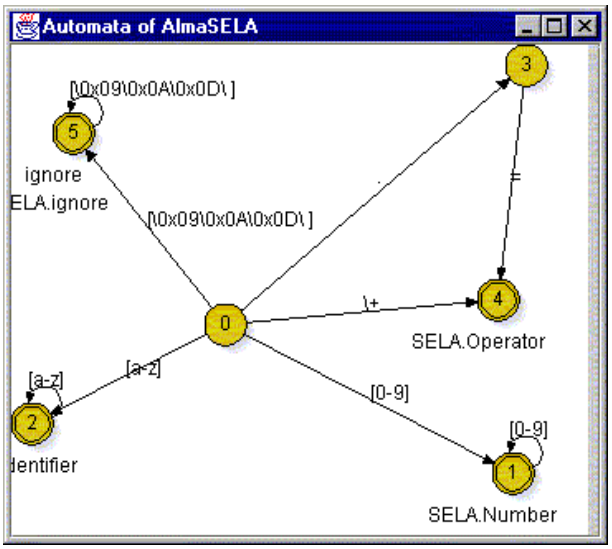


Figure 2: FSA visualizator

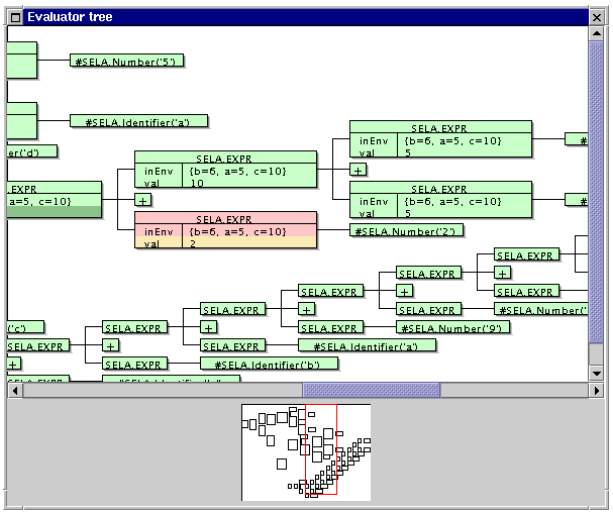


Figure 3: The snapshot of semantic tree animator

only in a very restricted manner. In many GBSs the transformation of the representation to a context-free grammar makes the various analyses of properties feasible. In others the ability of context-free grammars to represent infinite languages with a finite set of production rules is exploited. Sometimes, a problem can be solved simply by converting the representation to a context-free grammar since the appropriate tools already exist. Why is the study of GBSs so important? The theory of grammars is well-defined and described in many books, where different examples and solutions are presented [1, 32]. In our research we discovered that grammars can be, and already are, used as a kernel part of many different practical systems. The fact that grammars are so well-defined is an advantage for developers, because they can use that knowledge, and the already well tested solutions, to build their own. The main problem of using grammars as part of a solution is to identify those problems that have grammatical nature (can be solved with grammars) and to convert their presentation in the form of grammars.

One may ask why traditional programming applications such as compilers are excluded from the above definition since it is clear that these applications heavily depend on grammars. Actually, their whole construction is based on grammars. Grammars have been used in this area since their invention and other ad-hoc approaches were mainly superseded by grammars a long time ago. In this case we simply do not have other options. Therefore, talking about grammar-based compiler would be awkward. On the other hand, using grammars in other application areas can be regarded as an alternative and novel approach with clearly defined benefits. In this case the noun qualifier “grammar-based" is really appropriate.

Our longstanding interest in grammars inspired us to start collecting information about their different practical application areas, such as:

– *Software engineering*, where syntax definition occurs in various software development processes – in the form of rapid prototyping [28, 8], the modeling flow and constrains of collaborating software components [17], and many others [3, 15].

– *Evolutionary computation* is the study of computational systems that uses ideas from natural evolution and adoption to search the solution space. One of the research fields of evolutionary computations is grammatical evolution [26].

– *Information theory* comprises a vast range of diverse scopes. So far, our observations noticed grammars involved in encoding methods [2, 25], programming compaction [4] and grammatical inference [19]. Other grammar-dependent software in information theory are under investigation.

– *Neural networks* bring grammars into use with grammatical description of neural networks topology [14, 6, 10].

– *Data representation* architecture uses special kinds of grammars for communicating business data among very diverse systems. The particular technology considered is XML [29].

– Other areas of Computer Science (speech recognition, data mining, syntactical pattern recognition, etc).

Applications of grammars are found even in areas outside computer science, such as organizational science [27] and mechanical engineering [33].

Until now GBSs have been studied only for particular problems (e.g. compression) without any general comparison and classification. The only attempt, to the best of our knowledge, is a recent work described in [15] where authors coined the word *grammarware*. To quote their definition *"Grammarware comprises grammars and all grammar-dependent software, i.e. software artifacts that directly involve grammar knowledge."* Their definition classified compilers, program analysis tools, program transformation tools, application generators, weaving tools, CASE tools as grammarware. Their definition, is in some, sense more restricted than ours and includes just GBSs from those areas of software engineering where grammars appear in an explicit form. Sentences produced by this grammar are always computer programs. Our definition is much broader since we are interested in GBSs for application areas that are even outside computer science (e.g. organizational science). On the other hand, in our definition grammars and sentences produced by this grammar can be expressed implicitly (e.g. in GOOD [17] a sentence is a sequence of method calls during the execution of an application program).

4 Grammar-Based Systems: Examples

GBSs can be found in different application areas such as: software engineering, evolutionary computations, information theory, neural networks, data mining, syntactical pattern recognition, and data representation. In this section some of our own applications, as well as other representative applications of grammars, are presented and their benefits are stressed in more detail. Examples clearly show how grammars and sentences generated by a grammar can be used to describe various structured artifacts. Moreover, various possibilities exist for using GBSs. Examples include descriptions of GBSs where sentences generated by grammar appear in explicit or implicit manner.

4.1 Software Engineering

4.1.1 Grammatical Approach to Problem Solving

In [8, 28] the grammatical approach to problem solving (GAPS) is presented. It is based on the following steps:

– describe the syntax of the problem (the structure of the classes that characterize problem domain), deriving the context-free grammar from the conceptual class diagram,

– describe the semantics of the problem (the meaning of the classes in the problem domain), associating attributes to every concept derived from the use cases and operational diagrams,

– generate a rapid prototype of the system, using a compiler generator and the attribute grammar obtained in the two previous steps.

Only the first step is explained for the purpose of this paper. A detailed explanation of the above steps can be found in [8].

The role of non-terminal symbols in a context-free grammar is two fold. First, at a higher abstraction level non-terminal symbols are used to describe different concepts in the programming language (e.g. an expression or a declaration in a general-purpose programming language). On the other hand, at a more concrete level, non-terminal and terminal symbols are used to describe the structure of a concept (e.g. an expression consists on two operands separated by an operator symbol, or a variable declaration consists of a variable type and a variable name). Therefore, both the concepts and the relations between them, belonging to the specific problem domain, are captured in a context-free grammar. But, this is also true for the conceptual class diagram [30] which describes concepts in a problem domain and their relations. It is clear that both formalisms can be used for the same purpose and that some rough transformation from a conceptual class diagram to a context-free grammar and vice versa should exist. The transformation from a conceptual class diagram to a context-free grammar is depicted in Tables 1 and 2.

Classes can collaborate with more than just one class. For example, class A associates with classes B, C and D. In our approach, this collaboration is described with context-free grammar production $A \rightarrow B\,C\,D$. The sequence of non-terminal symbols on the right side of the production should be in natural order and depends on the collaboration of entities in a given problem domain.

As an example let's transform the conceptual class diagram in Fig. 4 to a context-free grammar. From this conceptual class diagram the following context-free grammar is obtained using transformation Tables 1 and 2.

```
VIDEO_STORE ::=  MOVIES CUSTOMERS
MOVIES      ::=  MOVIES MOVIE  |  MOVIE
MOVIE       ::=  title PRICE
CUSTOMERS   ::=  CUSTOMERS CUSTOMER |  epsilon
CUSTOMER    ::=  name RENTALS
RENTALS     ::=  RENTALS RENTAL  |  RENTAL
RENTAL      ::=  daysRented MOVIE
PRICE       ::=  new | child | reg
```

The next step of the grammatical approach is to write a detailed semantic description of the problem domain deriving the attribute grammar. The prototype of the system is

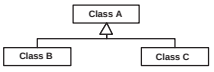
Association	Class diagram element	Grammar
Class		Class (non-terminal) attribute (terminal)
Association		A ::= B
Navigability		A ::= B
Generalization		A ::= B | C
Aggregation		A ::= B $(\neg\exists X \in N, X \Rightarrow B) \wedge X \neq A$
Composition		A ::= B

Table 1: From a conceptual class diagram to a context-free grammar

obtained after a straightforward transformation of attribute grammar specification to LISA specification. The following program describes a particular use of the system.

```
//entering movies into database
lion_king child
gone_with_the_wind reg
the_ring new
//customers and their rentals description
Andy 3 lion_king child
     2 gone_with_the_wind reg
Mary 3 the_ring new
```

What are the advantages of using grammars in this case? By transforming the conceptual class diagram to a context-free grammar a domain-specific language is obtained that describes the user interaction with the system. In this manner a rapid prototype is obtained and can be used whenever

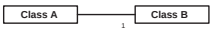
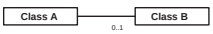
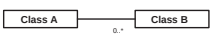
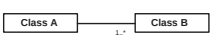
Cardinality	Class diagram element	Grammar
Multiplicity exactly one		A ::= B
Optional multiplicity		A ::= B | ϵ
Multiplicity [0..m]		A ::= MoreB MoreB ::= MoreB B | ϵ
Multiplicity many		A ::= MoreB MoreB ::= MoreB B | B

Table 2: Association multiplicity

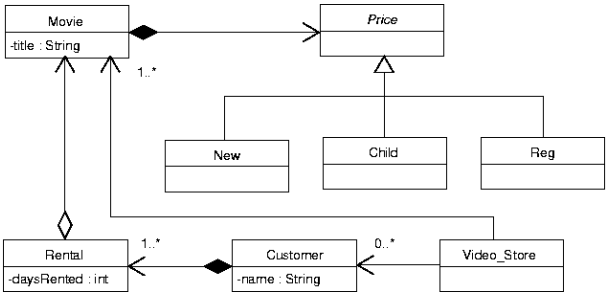


Figure 4: Conceptual Class Diagram for Video Store

the user's requirements are not well defined. Another benefit is that a conformability check of the conceptual class diagram is also possible.

4.1.2 Adaptive programming

Adaptive programming (AP) [18] is a subclass of aspect-oriented programming [13]. The stress is put on code tangling – for example, the required functionality is not always trivial to implement in existing applications when cross-cutting concerns exist. Adaptive programming offers a solution in the form of traversal specifications, in order to provide that additional functionality without modification of the existing code. These specify connections between objects as loosely as possible (called "structure-shy" programming).

The structure of the application class dictionary can be seen as a context-free grammar (Figure 5) from which an object graph may be derived, express all possible navigations through the code.

```
S ::= A  B | A S B
A ::= ...
B ::= C ...
```

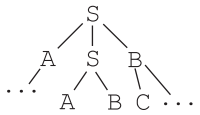


Figure 5: Class dictionary (CFG) and its object graph

The idea of adaptive programming is very general. The Demeter [18] language has been integrated with various object-oriented programming languages (the Demeter tool was successfully applied to Java). Demeter allows programmers to write the following specifications (see Fig. 5):

```
starting from object A, go to object C
via all objects with an attribute named "x".
```

to add arbitrary execution paths.

The problem of adaptive programming can also be solved using other techniques (visitor pattern). In comparison, the presented solution avoids code tangling, increases the programmer's productivity and consequently, it reduces error prone coding.

4.1.3 Other Approaches

In Grammar-Oriented Object Design (GOOD) [17] a context-free grammar is used to represent a set of all possible interactions (collaborations) for objects in a particular

cluster, in order to fulfill the domain goals. When a grammar is interpreted at run-time, a cluster will dynamically bind the collaborators to the collaborations. Hence, GOOD facilitates the creation of dynamically configurable components, which encapsulates volatile business rules. The rationale behind this is that creating and representing a model of solutions is more extensible, simpler and more scalable than just creating the single solution. Possible solutions are modeled with a meta-model and represented as a context-free grammar. If this grammar is available to the "users" at run-time, then they are able to customize the system's behavior. An example of a production rule in [17] using EBNF is:

```
ShoppingCartOperation ::=
  {AddItem | DeleteItem |
   SaveShoppingCart} CheckOut
```

Since the interaction of objects is obtained from use case diagrams that describe the functionality of a system, the author [17] called such a grammar a use case grammar. The author [17] in his work distinguishes two types of meta-models: the static (class diagram) and the dynamic (valid object interaction sequences) meta-model. The latter is described with a context-free grammar.

In [3] the correspondence between the feature diagram and the context-free grammar has been identified, where atomic features map to terminal symbols, composite features map to nonterminal symbols, and feature operators map to syntax operators. In domain analysis, feature diagrams are used to describe commonalities, variabilities and dependencies between variable properties in the application domain. By converting a feature diagram to a context-free grammar (FD2CFG), syntax tools can be applied to feature descriptions for free (e.g. validity of configuration corresponds to successful parsing).

The Free University of Amsterdam recently launched a project on Grammar Engineering - software engineering for grammars [15]. Topics included are grammar recovery, grammar implementation, and the application of grammars in software renovation. The more technical issues include concepts and technology for grammar-based software renovation factories, grammar adaptation, grammar documentation, grammar testing and many others.

4.2 Evolutionary Computations

Genetic programming (GP) is an evolutionary approach in which an evolving population consists of computer programs [16]. Each member of the population, a chromosome, represents a possible solution in the search space of all possible programs written in a *pre-selected programming language* (e.g. Lisp). Since the search space is too large it is restricted by the user-defined function set F and the terminal set T. The set T contains variables and constants and the set F functions that are a priori believed to be useful for the problem domain. For example, in the Santa Fe ant trail problem [16] from sets T = {(MOVE), (LEFT), (RIGHT)} and F = {IF-FOOD-AHEAD, PROGN2, PROGN3} the following solution (lisp program) can be evolved:

```
(IF-FOOD-AHEAD (MOVE)(PROGN3 (LEFT)(PROGN2
(IF-FOOD-AHEAD (MOVE)(RIGHT))(PROGN2 (RIGHT)
(PROGN2 (LEFT)(RIGHT))))(PROGN2
(IF-FOOD-AHEAD (MOVE)(LEFT))(MOVE))))
```

In [26] the concept of grammatical evolution (GE) has been introduced. GE is an evolutionary algorithm that can evolve programs in an *arbitrary language* [31]. The input to the GE is a BNF definition for the genotype-to-phenotype mapping process. For example, the following grammar can be used as input to Santa Fe ant trail problem:

```
0. CODE ::= LINE
1. CODE ::= CODE LINE
0. LINE ::= EXPR
0. EXPR ::= IF-STAT
1. EXPR ::= OP
0. IF-STAT ::= if (food-ahead()) EXPR else EXPR
0. OP ::= left()
1. OP ::= right()
2. OP ::= move()
```

The population consists of variable-length binary strings that determine which production rules from the grammar definition are used in a genotype-to-phenotype mapping process. The appropriate production rule is selected by using the following mapping function:

$rule =$ *(Integer value stored in a chromosome)MOD (number of production rules for the left most nonterminal)*

For example, the following chromosome (203 245 110 55 29 200 241 11 151 162 227 74) encodes the following left-most derivation

CODE $\Rightarrow^{203 MOD 2}$ *CODE LINE* $\Rightarrow^{245 MOD 2}$
CODE LINE LINE $\Rightarrow^{110 MOD 2}$ *LINE LINE LINE* $\Rightarrow$
EXPR LINE LINE $\Rightarrow^{55 MOD 2}$ *OP LINE LINE* $\Rightarrow^{29 MOD 3}$
move() LINE LINE $\Rightarrow \ldots \Rightarrow$
move() if (food-ahead()) move() else left() move()

What are the benefits of using grammars in GE? Obviously, GE is much more flexible than GP because it can produce a code in any language. Furthermore, in the GE closure problem, the generation and preservation of valid programs, does not exist. Other benefits come with the separation of the search and solution spaces because grammar enables the genotype-to-phenotype mapping process. This allows an unconstrained evolutionary search to be performed on simple variable-length binary strings. Moreover, new advances in genetic algorithms can be easily incorporated into GE or any new search algorithm operating on binary strings can be used.

4.3 Information Theory

The ability of a grammar to represent an infinite language with a finite set of production rules also makes grammars useful in compression algorithms. Grammar-based encoding (GBEnc) methods, such as derivation encoding [34], which represents a program by a sequence of grammar rules to derive it from the start symbol, have been proven

useful for compressing programs. For example, the program using the grammar from subsection 4.2 can be encoded as *1 1 0 1 2 0 1 2 1 0 1 2* by derivation encoding. The derivation tree is shown in Figure 6. It was shown in [2] that programs can be compressed to almost 10% of their original size. Another grammar-based compression algorithm is SEQUITUR [25] which constructs a context-free grammar for its input. The resulting grammar is capable of generating just one string, namely the original sequence. For example, the sequence $abcdbcabcd$ is represented by the following grammar

```
S ::= CAC
A ::= bc
C ::= aAd
```

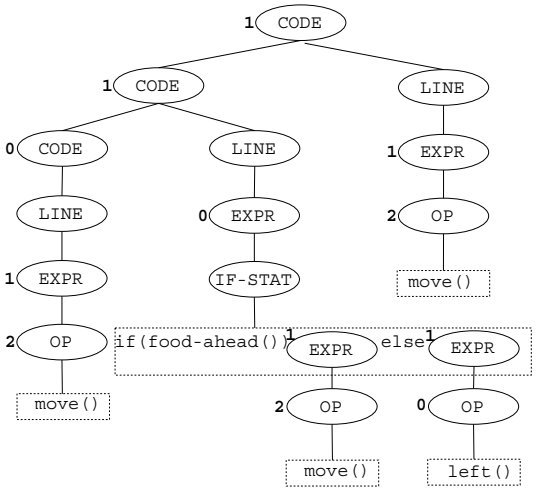


Figure 6: Derivation tree

The algorithm identifies the hierarchical structure (Figure 7) in sequences of symbols and uses that information for compression. By detection and elimination of redundancy it outperforms the standard compression techniques on very large or highly structured sequences. SEQUITUR preforms well on many practical problems such as DNA sequences and genealogical databases. It is also possible to use the system as a basis for generalization in grammatical inference [19]. Grammar-based techniques (e.g. [4]) have also been used in program compaction, which is a compression technique with an additional constraint - the compressed program has to be executable.

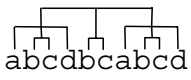


Figure 7: Hierarchical structure for grammar

4.4 Neural Networks

The selection of a suitable neural network topology is an important step in finding a good solution for the problem under investigation. Therefore, the search for suitable neural network architecture is a common task. Here we can use direct encoding, or the so-called grammatical encoding [14], where the architecture of the neural network is generated from its grammar description. The advantages of grammatical encoding are better scalability and the possibility of finding building blocks. This work is further elaborated in [6] where cellular encoding using graph grammars was proposed. The architecture, the weights, and the kind of sigmoids used by each neuron are encoded. Cellular encoding can be seen as a machine language for neural networks and can be used as a tool for designing neural networks.

In [10] an attribute grammar is used to specify classes of neural network structures with explicit representation of their functional organization. The approach is termed Network Generating Attribute Grammar Encoding (NGAGE). The specification of a neural network structure is extracted from the attributes of the root symbol and interpreted to produce a functional neural network. This neural network can be randomly initialized and trained afterwards. The NGAGE is specially usable in genetic programming, in the form of neural network representation, where each production rule (derivation subtree) corresponds to a meaningful structural component of the neural network. These characteristics of NGAGE can be used for genetic operators implementation, crossover and mutation. The other benefit of NGAGE is identical representation of different neural networks. The similarities and differences between them can be emphasized within a common framework.

4.5 Data Representation

The use of mark-up languages on the Web is indispensable. The *hypertext mark-up language* (HTML) is the best known example. In the last few years the *eXtensible Markup Language* (XML) has been introduced, as a mark-up language for uniform representation of data. It was originally meant as a format for transferring data over the Internet. Separation of data from its representation increased the standard applicability to other computing areas.

Although our perception is that compiler notation and mark-up language have little in common, the reality is quite different. The syntax of XML documents is conceptually similar to the meta-language (defined by BNF) of compilers. The analogy between compilers and mark-ups is shown in table 3.

Notation	**Compiler**	**Mark-up**
meta-notation	LISA, ASF+SDF	XML
syntax	context-free grammar	DTD, XML Schema

Table 3: Analogy between compiler and mark-up

The syntax of an XML document is defined by Document Type Definition (DTD). DTD defines document structure, elements, their attributes and types (see example be-

low). It uses the syntax of EBNF ('*', '+', '?', '|') to describe the syntactical structure of XML documents. Therefore, a DTD can be seen as an extended BNF (EBNF).

```
<!ELEMENT paper_collection (paper)*>
<!ELEMENT paper (title, author+, year,
                 published?)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT published EMPTY>
```

An example of an XML document, for the given DTD above, is written below. The *XML elements* from the DTD can be seen as non-terminal symbols in EBNF. *Empty elements* are without contents (element '<published>') - their meaning is in the position and attributes. *Non-empty elements* can contain other elements and textual content (element '<title>'). Textual content can be parallelized with terminal symbols in EBNF.

```
<paper_collection>
  <paper>
    <title>Grammatical Approach to
      Problem Solving</title>
    <author>Pedro Henriques</author>
    ...
    <year>2003</year>
    <published/>
  </paper>
  ...
</paper_collection>
```

Another similarity of mark-up languages and compilers is the building of a parser. The XML parser generator analyzes the source DTD and automatically generates a parser for XML documents which comply with the source DTD, which is also the case in compiler building tools (e.g. LISA [20]).

The disadvantage of XML against context-free grammar is in its lexical part. The textual content of XML elements can be either a generic string (denoted with '#PCDATA' in DTD) or enumeration of allowed values. This limitation is reduced with XML Schema, which offers richer notation for describing the textual content of the XML elements.

4.6 Other Applications

Due to page limitation, all GBSs can not be described in detail. The aim of the paper is to show a plethora of different research areas where grammars are proven to be useful. A short description of such systems in other areas follows.

A number of researchers have proposed ways to use grammar-based notation for expressing knowledge in the speech recognition process. In most cases CFG was used to generate or filter word transitions [24]. To improve semantic sentence recognition, the probabilistic LR parser has been used as well as stohastic CFG (SCFG) [12].

Data mining is an automated process of discovering knowledge from databases. Various data mining methods exist, among them are inductive logic programming and genetic programming. In [36] both approaches were integrated using logic grammars aiming to exploit the benefits of both approaches. Special rule learning has been developed where a grammar represents rules. Moreover, the grammar can be modified in order to learn rules.

Formal language theory has been successfully applied to pattern recognition problems [5] in which the patterns contain most of their information in their structure rather than in their numeric values. In order to make grammars more suitable for pattern recognition the concept of context-free grammars have been extended to stochastic grammars [35] and fuzzy grammars [11].

The design of organizational and work processes is well defined. However, there is a lack of formal approaches in discovering new organizational processes. By using grammars, one may systematically search for solutions in process redesign, as well as for new solutions in process organization. Therefore, the process grammar [27] offers complementary solutions to the existing ones.

4.7 Concluding remarks on GBSs examples

It is important that GBSs are not studied in an isolated manner. In order to be able to make a general comparison of different GBSs and to classify them, we need to find their common and variable properties. The main reason for classification of GBSs is to identify differences among GBSs and to identify the representative examples of it. These classifications can be used by future developers to identify whether their system is solvable using a grammar-based approach. Developers can further use the classification to build their own system faster and more efficiently.

The following questions help to identify different dimensions of GBSs:

Q1 What is described with grammar G?

Q2 What is described with program P generated by language L(G)?

Q3 Is the representation of program P generated by language L(G) explicit?

Q4 Is the control flow from G $\rightarrow$ P or P $\rightarrow$ G? (Is the input to GBS defined by grammar or program?)

Q5 Why was GBS invented?

In table 4, the answers to some examples are given.

It is important for future application developers to notice that with a grammar-based approach their systems can benefit in several directions:

- system can become more general (e.g. GOOD [17]),
- system can be easier to develop (e.g. [3]),
- system's underlying representation can be more efficient (e.g. [10]).

	Q1	Q2	Q3	Q4	Q5
GAPS	conceptual class diagram (CCD)	user interaction with the system	yes	G → P	to obtain rapid prototype of the system
GOOD	interaction between objects	sequence of method calls executed by an application program	no	G → P	to extend generality of the system
FD2CFG	feature diagram (FD)	an instance of a system described by FD	yes	G → P	to check if an instance is a valid system by FD
GE	grammar of the target language used in GP	program written in a target language	no	G → P	to extend generality of the system
GBEnc	grammar of the target language	program written in a target language to be compressed	yes	P → G	for compression
SEQUITUR	grammar of the target sequence of symbols	sequence of symbols	yes	P → G	for compression
AP	connections between objects	structure of application class dictionary	no	G → P	to extend functionality of applications
NGAGE	neural network (NN) structure	a fully functional NN	no	G → P	to simplify NN representation for GP

Table 4: A comparison among different GBSs

This paper describes some of the representative examples of the above mentioned benefits. The main contribution of this paper therefore is:

- definition of grammar-based systems,
- identifying problems that can be solved with grammar-based approach,
- identifying benefits of grammar-based system, and
- popularizing grammar-based systems.

5 Conclusions and future work

Formal language theory has been applied to many practical applications. In addition to language description and implementation (original applications of grammars), grammars have been proven useful in many other areas. However, there is no particular research of systems (applications) in which grammar plays a vital role. In this paper such systems are introduced and defined as GBSs. The paper contains representative examples of GBSs in various areas of computer science, such as: software engineering, evolutionary computations, information theory, neural networks, and data representation.

Although the formal theory of grammar is well defined, there are still many research possibilities in the field of GBSs. We have noticed several unexplored areas in GBSs such as:

- *Classification of GBSs.* There is no classification of GBSs. We believe that this can be attained by questions similar to the ones proposed in section 4.7. However, further case studies of GBSs are required.
- *When to develop GBS?* No guidelines exist to show whether a particular problem should be solved with grammar knowledge.
- *GBSs patterns.* The remaining question is how to develop GBSs. Identifying patterns would improve and speed up the interest in developing GBSs.

In the future we plan to extend our research on GBSs. Our research will be focused on solving those problems presented in this paper and on finding other areas or problems that can be efficiently solved with the grammar-based approach. We want to show that problem definition using the formal approach (grammar) can increase the efficiency, reliability and generality of the solution.

6 Acknowledgements

We would like to thank Jeff Gray and anonymous referees for useful comments.

References

[1] A. V. Aho and J. D. Ullman. The theory of languages. *Mathematical Systems Theory*, 2(2):97–125, 1968.

[2] R. Cameron. Source encoding using syntactic information models. *IEEE Transactions on Information Theory*, 34(4):843–850, 1988.

[3] M. de Jonge and J. Visser. Grammars as feature diagrams. draft, Apr. 2002.

[4] W. S. Evans and C. W. Fraser. Grammar-based compression of interpreted code. *ACM Communications*, 46(8):61–66, 2003.

[5] K. Fu. *Syntactic Pattern Recognition and Applications*. Prentice-Hall, 1982.

[6] F. Gruau. *Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm.* PhD thesis,

Laboratoire de l'Informatique du Parallilisme, Ecole Normale Supirieure de Lyon, France, 1994.

[7] J. Heering and P. Klint. Semantics of programming languages: A tool-oriented approach. *ACM Sigplan Notices*, 35(3):39–48, Mar. 2000.

[8] P. Henriques, T. Kosar, M. Mernik, M. J. V. Pereira, and V. Žumer. Grammatical approach to problem solving. In *ITI 2003 : Proceedings of the 25th International Conference on Information Technology Interfaces*, pages 645–650. SRCE University Computing Centre, University of Zagreb, 2003.

[9] P. Henriques, M. V. Pereira, M. Mernik, M. Lenič, E. Avdičaušević, and V. Žumer. Automatic generation of language-based tools. In M. van den Brand and R. Laemmel, editors, *Electronic Notes in Theoretical Computer Science*, volume 65. Elsevier Science Publishers, 2002.

[10] T. Hussain and R. Browse. Attribute grammars for genetic representations of neural networks and syntactic constraints of genetic programming. In *AIVIGI'98: Workshop on Evolutionary Computation*, 1998.

[11] Y. Inagaki and T. Fukumura. On the description of fuzzy meaning of context-free languages. In L. Zadeh, K. Fu, K. Tanaka, and M. Shimura, editors, *Fuzzy Sets and Their Applications to Cognitive and Decission Processes*, pages 301–328. Academic Press, 1975.

[12] D. Jurafsky, C. Wooters, J. Segal, A. Stolcke, E. Fosler, G. Tajchman, and N. Morgan. Using a stochastic context-free grammar as a language model for speech recognition. In *Proc. ICASSP '95*, pages 189–192, Detroit, MI, 1995.

[13] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *Proceedings European Conference on Object-Oriented Programming*, volume 1241, pages 220–242. Springer-Verlag, 1997.

[14] H. Kitano. Designing neural networks by genetic algorithms using graph generation systems. *Complex Systems*, (4):461–476, 1990.

[15] P. Klint, R. Lämmel, and C. Verhoef. Towards an engineering discipline for grammarware. Draft, Aug. 2003.

[16] J. R. Koza. *Genetic Programming: On the Programming of Computers by Natural Selection*. MIT Press, 1992.

[17] K. Levi and A. Arsanjani. A goal-driven approach to enterprise component identification and specification. *Communications of the ACM*, 45(10):45–52, October 2002.

[18] K. J. Lieberherr. *Adaptive Object-Oriented Software: The Demeter Method with Propagation Patterns*. PWS Publishing Company, 1996.

[19] M. Mernik, M. Črepinšek, G. Gerlič, V. Žumer, B. R. Bryant, and A. Sprague. Learning context-free grammars using an evolutionary approach. Technical report, University of Maribor and The University of Alabama at Birmingham, 2003.

[20] M. Mernik, N. Korbar, and V. Žumer. LISA: A tool for automatic language implementation. *ACM SIGPLAN Notices*, 30(4):71–79, Apr. 1995.

[21] M. Mernik, M. Lenič, Enis Avdičaušević, and V. Žumer. Multiple Attribute Grammar Inheritance. *Informatica*, 24(3):319–328, Sept. 2000.

[22] M. Mernik, M. Lenič, E. Avdičaušević, and V. Žumer. LISA: An Interactive Environment for Programming Language Development. In N. Horspool, editor, *11th International Conference on Compiler Construction*, volume 2304, pages 1–4. Lecture Notes in Computer Science, Springer-Verlag, 2002.

[23] M. Mernik and V. Žumer. An educational tool for teaching compiler construction. *IEEE Transactions on Education*, 46(1):61–68, February 2003.

[24] R. Moore, F. Pereira, and H. Murveit. Integrating speech and natural-language processing. In *Proc. of the Speech and Natural Language Workshop*, pages 243–247, Philadelphia, PA, 1989.

[25] C. G. Nevill-Manning and I. H. Witten. Compression and explanation using hierarchical grammars. *The Computer Journal*, 40:103–116, 1997.

[26] M. O'Neill and C. Ryan. Grammatical evolution. *IEEE Transaction on Evolutionary Computations*, 5(4):349–358, August 2001.

[27] B. T. Pentland. Grammatical models of organizational processes. *Organization Science*, 6(5):541–56, 1995.

[28] M. V. Pereira, M. Mernik, T. Kosar, P. Henriques, and V. Žumer. Object-oriented attribute grammar based grammatical approach to problem specification. Technical report, University of Braga, Department of Computer Science, 2002.

[29] E. T. Ray. *Learning XML: Creating Self-Describing Data*. O'Reilly & Associates, Inc., 2001.

[30] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice-Hall, 1991.

[31] C. Ryan, J. J. Collins, and M. O Neill. Grammatical evolution: Evolving programs for an arbitrary language. In *Proceedings of the First European Workshop on Genetic Programming*, volume 1391 of *LNCS*, pages 83–95, Paris, 14-15 Apr. 1998.

[32] Salomaa, A. *Theory of Automata*. Pergamon Press, 1969.

[33] L. C. Schmidt and J. Cagan. Ggreada: A graph grammar-based machine design algorithm. *Research in Engineering Design*, 9:195–213, 1997.

[34] R. G. Stone. On the choice of grammar and parser for the compact analytical encoding of programs. *The Computer Journal*, 29(5):307–314, 1986.

[35] P. Swain and K. Fu. Stochastic programmed grammars for syntactic pattern recognition. *Pattern Recognition*, (4):83–100, 1972.

[36] M. L. Wong and K. S. Leung. *Data mining using grammar based genetic programming and applications*. Kluwer Academic Publishers, 2000.

Improved Error Recovery in Generated LR Parsers

Boštjan Slivnik and Boštjan Vilfan
University of Ljubljana
Faculty of Computer and Information Science
Tržaška 25, 1000 Ljubljana, Slovenia
bostjan.slivnik@fri.uni-lj.si
bostjan.vilfan@fri.uni-lj.si

Keywords: LR parsing, error recovery and reporting

Received: February 17, 2004

A new method for error recovery in LR *parsers is described. An error recovery routine based on this new method can be generated automatically by a parser generator as a part of an* LR *parser. Based on the result that a viable suffix from which the unread part of the input is derived can be computed in certain states of an* LR *parser, the new method uses the viable suffix to discard the erroneous part of the input and to synchronize the parser stack with the rest of the input afterwards. Thus it resembles a simple but efficient error recovery method used by* LL *and other predictive parsers. It is proved that all states suitable for this kind of error recovery can automatically be identified by a parser generator.*

Povzetek: članek opisuje okrevanje po napaki v LR analizatorjih.

1 Introduction

Compilers are programs that mostly process erroneous input. Robust error recovery and meaningful error reporting are therefore essential parts of any industrial-strength compiler.

Nowadays many compilers perform syntax analysis using an LALR parser (more precisely, $\mathrm{LA}(1)\mathrm{LR}(0)$ parser) that is generated automatically by a parser generator like *yacc*, *bison*, *JavaCUP*, etc. [3]. However, none of these parser generators uses any advanced method for error recovery and reporting mainly, because these methods are either (a) time consuming, or (b) require inspection of individual parser states and manual insertion of error recovery routines [4, 5].

LALR parser generators usually provide only a very simple mechanism for error recovery and none for error reporting. If a *yacc* generated parser is to recover after an error is encountered within a sentential form derived from a nonterminal A, a compiler writer should insert a production

$$A \longrightarrow \mathbf{error}\ \alpha$$

manually where **error** is a *yacc* reserved word [2, 3]. In case of an error, a parser abandons other productions expanding A, moves forward over the erroneous part of the input and discards it until a string which can be reduced to α is seen. A reduction to A is performed and thus the parser is *resynchronized*. However, "proper placement of **error** tokens in a grammar is a black art" [3].

The paper is organized as follows. Section 2 includes definitions of some basic elements of formal language theory and parsing. Following the brief outline of the new method in Section 3, the construction of the finite automaton used by the error recovery routine is described in Section 4. This is followed in Section 5 by (a) the algorithm for computing the viable suffix needed for error recovery and (b) the algorithm for computing a grammatical context of the erroneous part of the input. Examples and figures are given along the way.

2 Basic definitions

Standard terminology of formal language theory and parsing is assumed [4, 5]. Throughout the paper we assume that grammars are reduced (no useless or unreachable symbols) and \$-augmented (production $S' \longrightarrow \$S\$$ is added as the only production expanding the new start symbol S'). A string $\gamma \in V^*$ is a viable prefix (suffix) of G iff there exists a rightmost (leftmost) derivation $S \Longrightarrow^*_{G,\mathrm{rm}} \gamma u$ ($S \Longrightarrow^*_{G,\mathrm{lm}} u\gamma^R$).

The nondeterministic $\mathrm{LR}(k)$ machine N_k for the grammar $G = \langle V, T, P, S\rangle$ (where V contains both nonterminal and terminal symbols) is a finite (semi)automaton with the state set equal to I_k (the set of valid $\mathrm{LR}(k)$-items for G), an initial item $i_0 \in I_k$, and a mapping $\delta_N \colon I_k \times (V \cup \{\varepsilon\}) \longrightarrow 2^{I_k}$ [1].

The deterministic $\mathrm{LR}(k)$ (or $\mathrm{LR}(k)\mathrm{LA}(k')$) machine M_k for the grammar G is a finite (semi)automaton with a set of states $Q \subseteq 2^{I_k}$, an initial state $q_S \in Q$, and a mapping $\delta_M \colon Q \times V \longrightarrow Q$. If $\delta^*_M(q_S, \gamma) = q$ for some $q \in Q$ and $\gamma \in V^*$, then $[\gamma]$ denotes the set of equivalent viable prefixes leading from the initial state q_S to the state q. Furthermore, $[\gamma]$ uniquely determines q (and is thus just another name for q).

The $\mathrm{LR}(k)$ parser is a pushdown transducer $\langle M, \tau \rangle$ (or simply M). M denotes the deterministic pushdown automaton based on the deterministic $\mathrm{LR}(k)$ machine M_k, and τ denotes the output effect (a mapping of parser actions into grammar productions). States of M are the same as the states of M_k. The stack alphabet of M is a set of states of M_k. A configuration (or instantaneous description) of a parser M is represented as $\$\Gamma | u\$$, where $\Gamma \in Q^*$ and $u \in T^*$ denote the stack contents and the unread part of the input, respectively.

3 The outline of the new method

To relieve a compiler writer of the "black art" of proper placement of **error** productions, a better error recovery method is needed. Let us suppose that the input string uv' is being parsed with an $\mathrm{LR}(k)$ parser M. Starting with the initial stack contents Γ_0, the parser M performs the parser steps $\pi(u)$ corresponding to the derivation

$$\$\Gamma_0 | uv'\$ \Longrightarrow_M^{\pi(u)} \$\Gamma | v'\$ \tag{1}$$

and enters a configuration $\$\gamma | v'\$$ (note that γ and v' denote the stack contents and the unread part of the input, respectively). $\mathrm{LR}(k)$ parsers have *the correct prefix property* [4, 5], i.e., any string of terminals pushed on the parser stack is a prefix of some valid input. It follows from Derivation (1) that there exist derivations

$$S \Longrightarrow_{\mathrm{rm}}^* \gamma v_i \Longrightarrow_{\mathrm{rm}}^* uv_i \tag{2}$$

for various v_i and a single viable prefix γ where $\Gamma = \Gamma'[\gamma]$ (the stack contents Γ of parser M corresponds to the viable prefix γ of Derivations 2). Therefore, there exist leftmost derivations

$$S \Longrightarrow_{\mathrm{lm}}^* u\delta_i \Longrightarrow_{\mathrm{lm}}^* uv_i \tag{3}$$

for various viable suffixes δ_i.

Now suppose that $\$\Gamma | v'\$$ is an error configuration. In other words, v' cannot be derived from any viable suffix δ_i in any of Derivations (3). The idea, on which the new method is based, can be outlined as follows:

1. If there is only one viable suffix $\delta = X_1\hat{\delta}$ such that $\delta_i = \delta$ for any i in Derivations (3), and
2. if this particular δ is known in the error configuration $\$\Gamma | v'\$$,

then the parser should discard the next few tokens of input, resynchronize and continue parsing the string derived from $\hat{\delta}$.

If there exists a unique viable suffix δ, there are two approaches to discard the erroneous part of the input. The first is to skip everything until a string from $\mathrm{FIRST}_k(\hat{\delta}\$)$ is seen, and then to resynchronize by pushing the symbol X_1 on the stack. Using this approach, Derivation (1) can be extended with the derivation

$$\begin{array}{lcl} \$\Gamma | v'\$ & = & \$\Gamma | v_1'\hat{v}\$ \\ & \Longrightarrow_M^{\pi(v_1')} & \$\Gamma | \hat{v}\$ \\ & \Longrightarrow_M & \$\Gamma[\gamma X_1] | \hat{v}\$ \end{array}$$

where the steps denoted by $\pi(v_1')$ are used to skip the part of the input derived from symbol X_1. Thus, the string $\hat{v}\$$ is the longest suffix of $v'\$$ having a property that $(k\colon \hat{v}\$) \in \mathrm{FIRST}_k(\hat{\delta}\$)$.

The second approach is to skip everything until a string which can be reduced to X_2, the first symbol of $\hat{\delta}$, is read, and then to resynchronize by pushing X_1 and then X_2 on the stack. Formally, Derivation (1) is extended with the derivation

$$\begin{array}{lcl} \$\Gamma | v'\$ & = & \$\Gamma | v_1'v_2\hat{v}\$ \\ & \Longrightarrow_M^{\pi(v_1')} & \$\Gamma[\gamma X_1] | v_2\hat{v}\$ \\ & \Longrightarrow_M^{\pi(v_2)} & \$\Gamma[\gamma X_1][\gamma X_1 X_2] | \hat{v}\$ \end{array}$$

where the steps denoted by $\pi(v_1')$ are used to skip the part of the input derived from symbol X_1 (as in the case above) and where $\pi(v_2)$ is the parse of v_2, the correct part of the input, in M.

The parser using an error recovery routine based on either of the two strategies tries to skip as few symbols of the input string as possible. More precisely, there may be many different, but viable splittings of the input string v' either to strings v_1' and $\hat{v}$ (as in the first approach) or to strings v_1', v_2 and $\hat{v}$ (as in the second approach). However, the problem of splitting the string v' is beyond the scope of this paper.

Finally, if the viable suffix δ cannot be determined uniquely in the parser state $[\gamma]$, the parser removes one state at the time from the parser stack until a state with a unique viable suffix is at the top of the stack. Then, one of the approaches described above is applied.

4 The construction of the error recovery routine

Two conditions were set in the previous section that must be fulfilled if a parser is to recover from the error: (1) the viable suffix δ must be unique and (2) it must be known. In general, a viable prefix γ and thus a state $[\gamma]$ (a set of $\mathrm{LR}(k)$-equivalent viable prefixes) can have many corresponding viable suffixes δ_i. To identify states of an $\mathrm{LR}(k)$ parser suitable for performing error recovery we start with the following two definitions [6]:

Definition 1 *Let $N_k = \langle I_k^G, V, P_{N_k}, i_0 \rangle$ be a nondeterministic $\mathrm{LR}(k)$ machine for a grammar $G = \langle V, T, P, S \rangle$. A string of $\mathrm{LR}(k)$-valid items $i_0 i_1 \ldots i_n \in (I_k^G)^*$ is called a $\langle \gamma, k \rangle$-path if there exists a sequence $X_1, X_2, \ldots, X_n \in (V \cup \{\varepsilon\})$ so that $X_n \neq \varepsilon$ and $[i_{j-1} X_j \to i_j] \in P_{N_k}$ where $i = 1, 2, \ldots, n$.*

Definition 2 *$\langle \gamma, k \rangle$-paths ρ_1 and $\rho 2$, where $\rho_1 = i_0 i_1 \ldots i_n$ and $\rho_2 = i_0 i_1' \ldots i_m'$ are 0-equivalent iff $n = m$ and items i_j and i_j' differ only in lookahead strings for all $j = 1, 2, \ldots, n$ (i.e., if $i_j = [A \to \alpha \bullet \beta, x]$ and $i_j' = [A' \to \alpha' \bullet \beta', x']$, then $A = A'$, $\alpha = \alpha'$, and $\beta = \beta'$).*

The reason for defining 0-equivalence of $\langle\gamma, k\rangle$-paths becomes obvious with the following lemma, which establishes a relationship between viable prefixes and suffixes on the one hand and $\mathrm{LR}(k)$ machines on the other.

Lemma 1 *Any two* 0*-equivalent* $\langle\gamma, k\rangle$*-paths define the same viable suffix.*

PROOF: Any $\langle\gamma, k\rangle$-path $\rho = i_0 i_1 \dots i_n$ specifies a set of leftmost derivations all having the form

$$\begin{aligned}
A_1 \quad &\Longrightarrow_{\mathrm{lm}}^{p_1} && \alpha_1 A_2 \beta_1 \\
&\Longrightarrow_{\mathrm{lm}}^{\pi(\alpha_1)} && u_1 A_2 \beta_1 \\
&\Longrightarrow_{\mathrm{lm}}^{p_2} && u_1 \alpha_2 A_3 \beta_2 \beta_1 \\
&\Longrightarrow_{\mathrm{lm}}^{\pi(\alpha_1)} && u_1 u_2 A_3 \beta_2 \beta_1 \\
&\ \ \vdots \\
&\Longrightarrow_{\mathrm{lm}}^{p_m} && u_1 u_2 \dots u_{m-1} \alpha_m A_{m+1} \beta_m \beta_{m-1} \dots \beta_1 \\
&\Longrightarrow_{\mathrm{lm}}^{\pi(\alpha_1)} && u_1 u_2 \dots u_m A_{m+1} \beta_m \beta_{m-1} \dots \beta_1,
\end{aligned}$$

where $p_j = A_j \longrightarrow \alpha_j A_{j+1} \beta_j$ is the production of the j-th $\mathrm{LR}(k)$-item of ρ having the dot in the initial position at the far left (and A_{m+1} may be ε). As any change of lookahead strings in $\mathrm{LR}(k)$-items of ρ does not affect the leftmost derivations above, all $\langle\gamma, k\rangle$-paths which are 0-equivalent to ρ define the same viable suffix δ, where $\delta^R = A_{m+1} \beta_m \beta_{m-1} \dots \beta_1$.
□

When the traditional $\mathrm{LR}(k)$ parser enters the error configuration $\$\Gamma \mid v'\$$, the error is recognized because no action is specified for q and $x = k{:}\ v'$, where $\Gamma = \Gamma' q$, i.e., $\mathtt{ACTION}(q, x) = \mathit{error}$. But as $\mathrm{LR}(k)$ parsers have the correct prefix property, the first $(k-1)$ symbols of the lookahead buffer are correct — otherwise the error would have been detected earlier.

The first step is to identify all states $[\gamma]$ with the property that for any $\gamma' \in [\gamma]$, all $\langle\gamma', k\rangle$-paths ending with an item $(k-1)$-active for $(k-1){:}\ v'$ are 0-equivalent (an item $[A \to \alpha X \bullet \beta, y]$ is k-active for x if and only if $x \in \mathrm{FIRST}_k^G(\beta y)$). In other words, the stack contents of the $\mathrm{LR}(k)$ parser and the first $(k-1)$ symbols in the lookahead buffer must define the viable suffix uniquely (remember that in error configuration $\$\Gamma \mid v'\$$ the k-th symbol of v' is erroneous and thus not useful for error recovery). To do so, we change the focus to the nondeterministic $\mathrm{LR}(0)$ machine N_0.

Definition 3 *An* $\mathrm{LR}(0)$*-item* $[A \to \alpha \bullet \beta]$ *of* N_0 *is relevant for state* $[\gamma]$ *of the deterministic* $\mathrm{LR}(k)$ *machine for* G *if and only if, for all* $\gamma' \in \gamma$ *and* $i' = [A \to \alpha \bullet \beta, y] \in [\gamma]$*, all* $\langle\gamma', k\rangle$*-paths* $\rho = \rho' i'$ *are* 0*-equivalent.*

During the parser construction, we compute a directed graph G_N with a set of vertices V_N and a set of edges E_N defined as

$$\begin{aligned}
V_N \quad = \{ \quad & \langle [A \to \alpha \bullet \beta], q \rangle \\
& | \exists q \in Q_M, y \in T^* {:}\ [A \to \alpha \bullet \beta, y] \in q\}
\end{aligned}$$

and

$$\begin{aligned}
E_N \quad = \{ \quad & \langle \langle i_1, q_1 \rangle, \langle i_2, q_2 \rangle \rangle | \exists X \in V \cup \{\varepsilon\} : \\
& [q_1 X \to q_2] \in P_M \wedge [i_1 X \to i_2] \in P_{N_0}\},
\end{aligned}$$

respectively. It is derived from the graph of the nondeterministic $LR(0)$ machine by (1) replicating each $LR(0)$-item as many times as there are states in M in which an LR(k) item with the corresponding core appears, and (2) erasing edge labels. Thus every path in G_N starting with $\langle [S' \to \bullet \$ S \$], q_S \rangle$ has its corresponding path in N_0 (and vice versa).

As an example, consider the following \$-augmented LALR grammar G_{ex} with productions

$$\begin{aligned}
&S' \longrightarrow \$S\$,\ S \longrightarrow AB, \\
&A \longrightarrow aA,\ A \longrightarrow \varepsilon, \\
&B \longrightarrow Bb,\ B \longrightarrow b.
\end{aligned}$$

The LALR machine for the grammar G_{ex} as constructed by a modified version of *bison* is shown in Figure 1. The graph G_N for the grammar G_{ex} is shown in Figure 2.

The irrelevant vertices of G_N (i.e., vertices $\langle i, q\rangle$ where i is irrelevant for q) can be identified by the following simple algorithm:

1. Compute the set of all conflicting vertices, i.e., those with at least two different predecessors in the same state:

$$\begin{aligned}
\bar{V}_N^{(1)} \quad = \{ v | \quad & v = \langle [A \to \bullet \beta], q \rangle \wedge \\
& \langle \langle i_1, q \rangle, v \rangle, \langle \langle i_2, q \rangle, v \rangle \in E_N \wedge \\
& i_1 \neq i_2 \}
\end{aligned}$$

   (Different predecessors from different states represent no problem as the state itself helps determining the right path.)

2. Compute the set of successors of conflicting vertices:

$$\bar{V}_N^{(2)} \quad = \quad \{v | v \text{ is-reachable-from } v' \in \bar{V}_N^{(1)}\}$$

In Figure 2 the irrelevant vertices are shaded while in Figure 1 they are closed between double lines.

We define *the skeleton automaton* U with the set of states

$$Q_U = \{i | \langle i, q \rangle \in V_N \setminus (\bar{V}_N^{(1)} \cup \bar{V}_N^{(2)})\},$$

alphabet Q_M, and the mapping $\delta_U{:}\ Q_U \times Q_M \longrightarrow Q_U$ defined as

$$\begin{aligned}
&\delta_U(i, q') = i' \Longleftrightarrow \\
&\quad \langle \langle i', q' \rangle, \langle i, q \rangle \rangle \in E_N \wedge \\
&\quad \langle i, q \rangle, \langle i', q' \rangle \in V_N \setminus (\bar{V}_N^{(1)} \cup \bar{V}_N^{(2)}).
\end{aligned}$$

The skeleton automaton is just a compact representation of the graph G_N with irrelevant vertices removed. The skeleton automaton for the grammar G_{ex} is shown in Figure 3. The aforementioned modification of *bison* makes

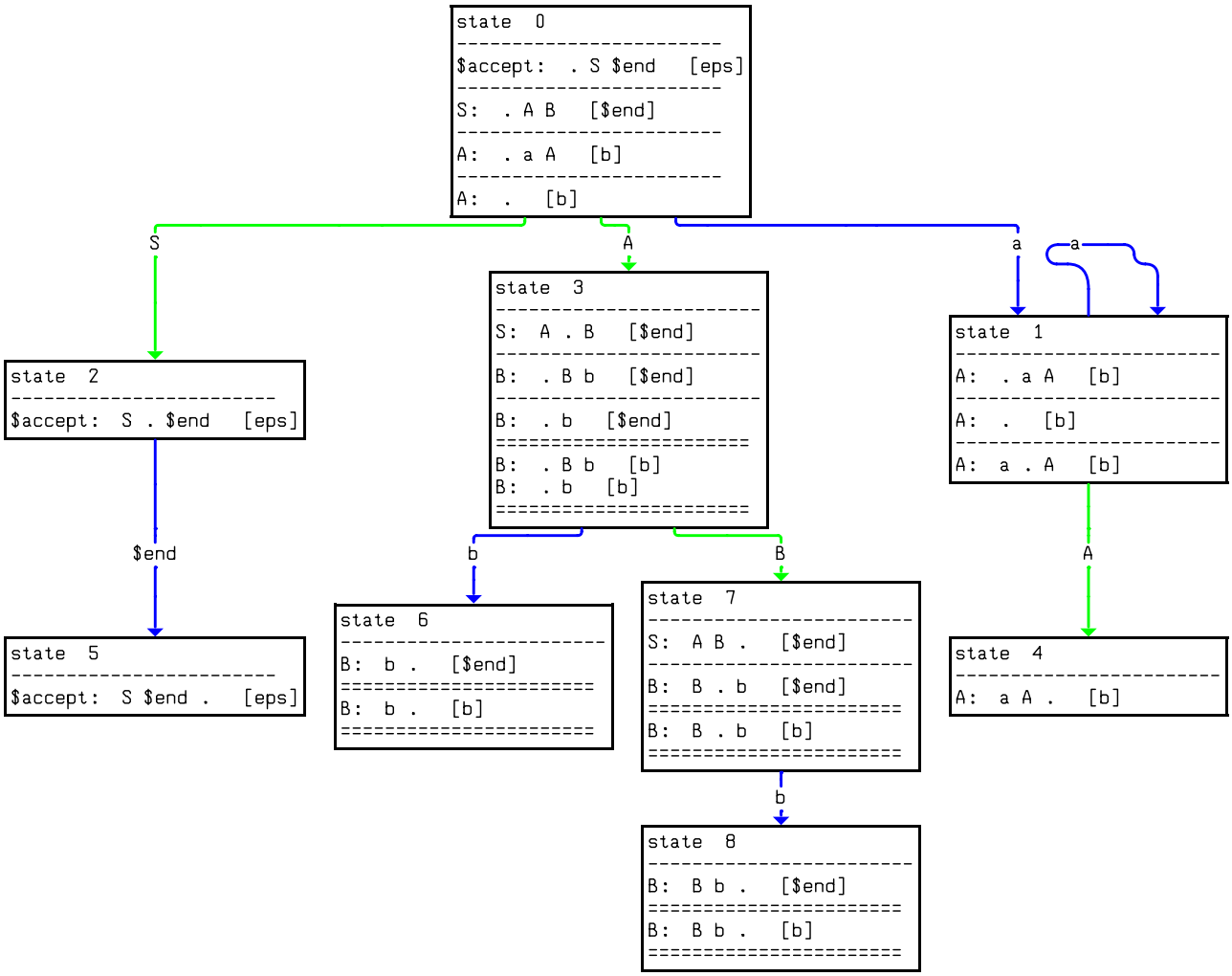


Figure 1: *Bison*-generated LALR machine for the grammar G. Note the different grammar augmentation: instead of using production $S' \longrightarrow \$S\$$, *bison* uses a production $\$_{\text{accept}} \longrightarrow S\$$.

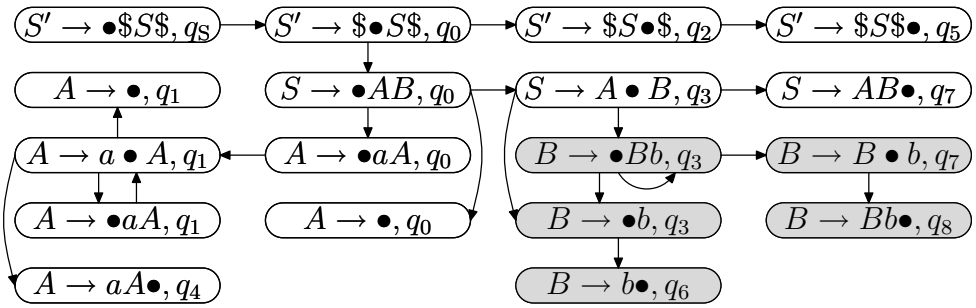


Figure 2: Graph G_N for the grammar G_{ex}.

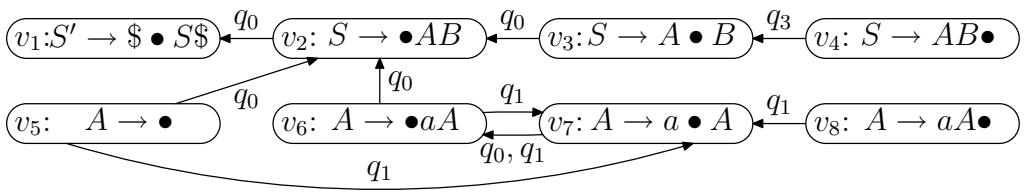


Figure 3: Skeleton automaton U for the grammar G_{ex}.

bison capable of computing the skeleton automaton — *bison*-generated skeleton automaton for the same grammar (although differently augmented) is shown in Figure 4.

(the modification of *bison* makes *bison* capable of computing the skeleton automaton as described below).

If the $\mathrm{LR}(k)$ parser is to start the error recovery process in state q and with the string x in the lookahead buffer, it should be able to select the right vertex of the skeleton automaton U. Hence, apart from the skeleton automaton, the parser must contain the table ERROR, which maps the topmost state and the first $(k-1)$ symbols of the lookahead buffer to a vertex of U:

$$\texttt{ERROR}\colon Q \times T^{*(k-1)} \longrightarrow V_N.$$

Construction of the table ERROR is straightforward. To compute the value of $\texttt{ERROR}(q, x)$, apply the following procedure:

1. If there exists a state of the skeleton automaton U corresponding to an item i where $\langle i, q\rangle \in V_N \setminus (\bar{V}_N^{(1)} \cup \bar{V}_N^{(2)})$ and all items of the core of the state q which are $(k-1)$-active for x map to i, i.e.,

   $$\forall [A \to \alpha \bullet \beta, y] \in \mathrm{Core}(q)\colon \\ x \in \mathrm{FIRST}_{k-1}(\beta y) \Longrightarrow [A \to \alpha \bullet \beta] = i,$$

   then $\texttt{ERROR}(q, x) = i$. Otherwise, the value of $\texttt{ERROR}(q, x)$ is undefined.

   (The set $\mathrm{Core}(q)$ contains either all items $[A \to \alpha \bullet \beta, y]$ where $\alpha \neq \varepsilon$ if $q \neq q_S$ or the item $[S \to \bullet\$S\$, \varepsilon]$ if $q = q_S$.)

2. If there exists exactly one node $\langle i', q\rangle \in V_N \setminus (\bar{V}_N^{(1)} \cup \bar{V}_N^{(2)})$ where $\delta_U(i', q) = i$ and there exists an item $[A \to \alpha \bullet \beta, y] \in q$ so that $x \in \mathrm{FIRST}_{k-1}(\beta y)$, then set $\texttt{ERROR}(q, x) = i'$ and repeat Step 2; otherwise, terminate.

   In other words, make the path leading from q to q_S as long as possible, but keep it unique in respect with the first $(k-1)$ symbols of the lookahead buffer.

Table 1 shows the ERROR table for the grammar G_{ex}.

LR STATE		SKELETON STATE
S		$[S' \to \bullet\$S\$]$
0	v_2:	$[S \to \bullet AB]$
1	v_7:	$[A \to a \bullet A]$
2	v_9:	$[S' \to \$S \bullet \$]$
3	v_3:	$[S \to A \bullet B]$
4	v_8:	$[A \to aA\bullet]$
5	v_{10}:	$[S' \to \$S\$\bullet]$
6		undefined
7	v_4:	$[S \to AB\bullet]$
8		undefined

Table 1: The ERROR table for grammar G_{ex}.

5 Computing the context of the syntax error

As mentioned at the end of Section 2, not every state of $\mathrm{LR}(k)$ parser is suitable for error recovery. If an error is detected in the state where error recovery cannot start, i.e., in the error configuration $\$\Gamma \,|\, v'\$$ where $\Gamma = \Gamma' q$, $x = k\colon v'$ and $\texttt{ERROR}(q, x) = \perp$, then the $\mathrm{LR}(k)$ parser must remove the topmost state from the stack and repeat the spawning of the error recovery. But as the lookahead string x in that state is no longer available, the parser must push the appropriate vertex of the skeleton automaton together with the at the time when the state itself.

More precisely, the parser stack should not contain just parser states, but pairs consisting of a parser state and a vertex of the skeleton automaton which is to be used if an error occurs. Hence, whenever the state q is pushed onto the stack (as a result of either shift or reduce action), it should be pushed as a pair $\langle q, \perp\rangle$. In the next step, before checking the action table and deciding on the next action, the parser must check the table ERROR and correct the value of the second component of the topmost pair on the stack.

The algorithm for computing the context in which a syntax error occurs is presented in Figure 5 (δ_U is a transition function corresponding to the set of rewriting rules P_U of skeleton automaton U). It starts at the top of the stack and proceeds downward. It produces a list of $\mathrm{LR}(0)$-items $i_1 i_2 \ldots i_m$ where $i_j = [A_j \to \alpha_j \bullet A_{j+1}\beta_j, y_j]$, which determine the derivation

$$\begin{aligned} A_1 \quad &\Longrightarrow^{i_1}_{\mathrm{lm}} \alpha_1 A_2 \beta_1 \\ &\Longrightarrow^{i_2}_{\mathrm{lm}} \alpha_1\alpha_2 A_3 \beta_2 \beta_1 \\ &\vdots \\ &\Longrightarrow^{i_m}_{\mathrm{lm}} \alpha_1\alpha_2 \ldots u_{m-1}\alpha_m A_{m+1}\beta_m\beta_{m-1}\ldots\beta_1 \end{aligned}$$

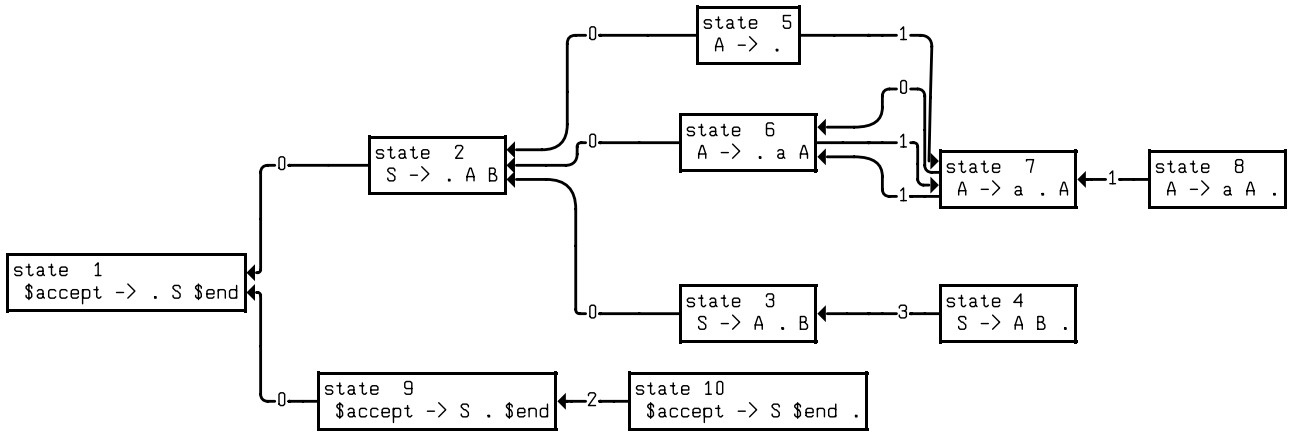


Figure 4: *Bison*-generated skeleton automaton U for the grammar G_{ex}.

$$
\begin{array}{l}
context\ stack\ i \mid i == [S' \to \$ \bullet S\$] \ = \ \varepsilon \\
\\
context\ stack@((q,\_):st)\ [A \to \bullet\beta] \ = \ ctx \circ [A \to \bullet\beta] \\
\qquad \text{where } i \ = \ \delta_U([A \to \bullet\beta], q) \\
\qquad\qquad ctx \ = \ context\ stack\ i \\
\\
context\ stack@((q,\_):st)\ [A \to \alpha X \bullet \beta] \ = \ ctx \circ [A \to \alpha X \bullet \beta] \\
\qquad \text{where } ctx \ = \ context\ st\ [A \to \alpha \bullet X\beta]
\end{array}
$$

Figure 5: Algorithm for computing the viable suffix.

and the viable suffix $\beta_1^R\beta_2^R\ldots\beta_m^R$. Hence, we can write down the following theorem:

Theorem 1 *If any two $\langle\gamma', k\rangle$-paths ending with items which are $(k-1)$-active for x, are 0-equivalent for each $\gamma' \in [\gamma]$, then a viable suffix δ_i in derivation (3) can be computed from the stack contents $\Gamma = \Gamma'[\gamma]$ in the parser configuration $\$\Gamma|v\$$ for any $v = xv'$.*

A parse of erroneous string $aacbb$ is shown in Tables 2 and 3. Both possible solutions mentioned in Section 2 are shown. In Table 2, the erroneous part of the input is discarded until the string $b \in \text{FIRST}_1(AB\$)$ is found in the lookahead buffer. The resulting stack contents after error recovery is performed is therefore $\$[\$][\$a][\$aa][\$aaA]$. This is a simple but efficient solution.

In Table 3, however, the string bb is reduced to the second symbol of the viable suffix $AB\$$, namely B. Hence, the resulting stack contents is $\$[\$][\$A][\$B]$. Finding and reducing the substring bb can be performed in the same way as by parsers generated by existing LALR parser generators if **error** productions are used.

Finally, the list of items

$$
\begin{array}{l}
[S' \to \$ \bullet S\$], [S \to \bullet AB], \\
[A \to \bullet aA], [A \to a \bullet A], \\
[A \to \bullet aA], [A \to a \bullet A]
\end{array}
$$

	STACK	INPUT
1.	$\$(q_0, v_2)$	$aacbb\$$
2.	$\$(q_0, v_2)(q_1, v_7)$	$acbb\$$
3.	$\$(q_0, v_2)(q_1, v_7)(q_1, v_7)$	$cbb\$$
	$\Rightarrow$ *context* returns	
	$[S' \to \$ \bullet S\$]$	
	$[S \to \bullet AB]$	
	$[A \to \bullet aA]$	
	$[A \to a \bullet A]$	
	$[A \to \bullet aA]$	
	$[A \to a \bullet A]$	
	yielding viable suffix $(AB\$)^R$:	
4.	$\$(q_0, v_2)(q_1, v_7)(q_1, v_7)(q_4, v_8)$	$bb\$$

Table 2: A trace of parsing with error recovery: erroneous part of the input is discarded until $b \in \text{FIRST}_1(AB\$)$ is seen.

	STACK	INPUT
1.	$\$(q_0, v_2)$	$aacbb\$$
2.	$\$(q_0, v_2)(q_1, v_7)$	$acbb\$$
3.	$\$(q_0, v_2)(q_1, v_7)(q_1, v_7)$	$cbb\$$
	$\Rightarrow$ $context$ returns	
	$[S' \rightarrow \$ \bullet S\$]$	
	$[S \rightarrow \bullet AB]$	
	$[A \rightarrow \bullet aA]$	
	$[A \rightarrow a \bullet A]$	
	$[A \rightarrow \bullet aA]$	
	$[A \rightarrow a \bullet A]$	
	yielding viable suffix $(AB\$)^R$:	
4.	$\$(q_0, v_2)(q_3, v_3)(q_7, v_4)$	$\$$

Table 3: A trace of parsing with error recovery: erroneous part of the input is discarded until bb derived from B in $AB\$$ is reduced.

represents a particularly good starting point for printing out helpful error messages because it provides the compiler writer with the exact grammatical context within which the error occurred.

6 Conclusion

The presented method works with both, canonical $\mathrm{LR}(k)$ parsers as well as $\mathrm{LA}(k)\mathrm{LR}(k')$ parsers. The error recovery routine does not slow down or influence the parser until it encounters the first error, and it can be generated automatically. Besides, it has two main benefits: (a) a compiler writer needs not add any additional productions to the grammar and (b) it is a good starting point for meaningful error reporting. However, the generation of parser is slower and the generated parser is larger.

References

[1] A. V. Aho and J. D. Hopcroft. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[2] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques and Tools*. Addison-Wesley series in Computer Science. Addison-Wesley, 1986.

[3] J. Levine, T. Mason, and D. Brown. *Lex & Yacc*. O'Reilly & Associates, 1992.

[4] S. Sippu and E. Soisalon-Soininen. *Parsing Theory, Volume I: Languages and Parsing*, volume 15 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1988.

[5] S. Sippu and E. Soisalon-Soininen. *Parsing Theory, Volume II: LR(k) and LL(k) Parsing*, volume 20 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1990.

[6] B. Slivnik. *Kombinacija Knuthovega in Lewis-Stearnsovega sintaksnega analizatorja z minimalno uporabo Knuthove analize*. PhD thesis, University of Ljubljana, Ljubljana, Slovenia, 2003.

Informational Design of Conscious Entities

Anton P. Železnikar
Volaričeva ul. 8, Ljubljana, SI–1111
s51em@hamradio.si

Keywords: definitions, informational design, informational consciousness, informational methodology, metaphysicalistic organization, thesaural approach

Received: June 8, 2003

Thesaural approach to entities' metaphysicalistic organization is shown within the design of an informational consciousness system.

Besides phenomenal, psychological, biological, quantum-mechanical, artificial, and other possible definitions of consciousness, the informational consciousness *(IC) is outlined as a major autonomous subsystem necessary in any conscious system. It is impossible to imagine a kind of the beyond to the informational phenomenalism in brain, nature, cosmos, and artifact. IC is founded formally by its own informational axiomatism, formulas, formula systems, schemes, scheme systems, graphs, and other constructive means and, within this theory and implementation possibilities, remains clearly scientifically disciplined. More actually, IC systems can be designed by the proposed informational methodology and implemented within existing complex computer systems and informational nets.*

Povzetek: članek opisuje informacijsko zasnovo zavestnih entitet.

1 Definitions of different sorts of consciousness

A precise and exhaustive definition of consciousness does not exist at all. Definitions can emerge according to the research of brain, mind, cognition, emotions, motivation, and other ingredients of the consciousness complex. There are loose attempts stating a few commonly accepted principles within the concept of consciousness. Definitional distinguishing among philosophical consciousness, artificial consciousness, phenomenal consciousness, psychological consciousness, biological consciousness, quantum-mechanical consciousness, informational consciousness, and other sorts of consciousness is possible.

Definitions of consciousness can substantially differ in different languages. The philosophical background dictates the inclination to such or another definition together with different views of research and the emerging science of consciousness. Let us make a short list of possible definitions explicitly concerning consciousness of such or another sort. In a way, major consciousness components like cognition and emotions should be incorporated in the rough forms of definition.

Instead with definitions we start with the assumptions of the following sort.

Assumption 1 *From the research point of view different sorts of* consciousness *can be distinguished:*

InC: individual (human, animal) consciousness,
BC: biological consciousness,
PhC: phenomenal consciousness,
PhiC: philosophical consciousness (phenomenology),
PsC: psychological consciousness,
SC: social consciousness,
LC: local consciousness,
GC: global consciousness,
CC: cosmological consciousness,
QtC: quantum-theoretical consciousness,
AC: artificial consciousness,
IC: informational consciousness,
AIC: artificial informational consciousness, etc.

These sorts of consciousness represent the majority of concepts known nowadays. □

Assumption 2 Consciousness *concerns an object or a complex of objects and emerges out of the* subconscious *or* unconscious *background, being the possible* informational space. *The substance of any sort of consciousness is informational and, within the informational, consciousness-like components can emerge being organized consciously.* □

This assumption does not deviate essentially from known definitions of consciousness. As an exception the *informational view* of consciousness may be understood which, however, seems to be evident.

Assumption 3 Consciousness *is generally regarded to be a* conscious system *with abilities or* components *representing conscious properties. One of the major system components is* self-awareness *or* self-consciousness, *the ability to perceive the component by itself. Perception as a conscious component perceives the relation between component and component environment.* □

To repeat: Informational consciousness seems to be the essential component or property of any consciousness system. We experience how the phenomenal consciousness — the consciousness in the living human brain — has its

evident informational roots, for instance, in pure linguistic experience, where memory performs as an associative storage of learned and experienced information. Phenomenology in philosophy performs as a literary information and, in this view, as informational phenomenology.

2 Informational consciousness system

Instead of consciousness a more appropriate term will be introduced, called *conscious system.* For a general denotation of an informational formula system, symbol Φ will be used. Conscious system is a formula system concerning consciousness as a particular informational entity, denoted by $\mathfrak{z}$. Thus, $\Phi\lceil\mathfrak{z}\rceil$ will symbolize the conscious system at the first glance, not expressed informonically yet. In general, it is adequate to use informational systems instead of properties like consciousness, subconsciousness, perception, cognition, emotions, and so on. A conscious system can unite such kind of subsystems organizationally. The system view of informational entities approaches also the understanding of informons and entropons as complex conscious and subconscious entities representing the named informational entities.

Assumption 4 Conscious system, *denoted naively (primitively) by* $\Phi\lceil\mathfrak{z}\rceil$, *is a system of major subsystems, where* $\mathfrak{z}$ *stands for consciousness, that is,* $\mathfrak{z} \rightleftharpoons \mathfrak{c}_{\text{consciousness}}$, $\mathfrak{s}_{\text{subconsciousness}}$ *for subconsciousness,* $\mathfrak{u}_{\text{unconsciousness}}$ *for unconsciousness,* $\mathfrak{c}_{\text{cognition}}$ *for cognition,* $\mathfrak{e}_{\text{emotions}}$ *for emotions,* $\mathfrak{m}_{\text{motivation}}$ *for motivation,* $\mathfrak{h}_{\text{homeostasis}}$ *for homeostasis,* $\mathfrak{b}_{\text{behavior}}$ *for behavior, etc.* $\Phi\lceil\mathfrak{z}\rceil$ *is meant to be a circular, component-distributed, serial-parallel system of components, tied informationally by common operands.* □

In the next step of theory development, the informonic view has to be considered to explicate the complex and *holistic* nature of a conscious system. An informon $\underline{\boldsymbol{\alpha}}$ always appears together with its adequately named entropon $\overline{\boldsymbol{\alpha}}$. They build the so-called α-informational space, denoted by $(\underline{\boldsymbol{\alpha}}, \overline{\boldsymbol{\alpha}})$. Or, consequently, recursively,

$$\underline{\boldsymbol{\alpha}},\ \overline{\boldsymbol{\alpha}} \quad \text{delivers} \quad (\underline{\boldsymbol{\alpha}}, \overline{\boldsymbol{\alpha}});$$
$$\underline{(\underline{\boldsymbol{\alpha}}, \overline{\boldsymbol{\alpha}})},\ \overline{(\underline{\boldsymbol{\alpha}}, \overline{\boldsymbol{\alpha}})} \quad \text{delivers} \quad \Big(\underline{(\underline{\boldsymbol{\alpha}}, \overline{\boldsymbol{\alpha}})}, \overline{(\underline{\boldsymbol{\alpha}}, \overline{\boldsymbol{\alpha}})}\Big);$$
$$\underline{\Big(\underline{(\underline{\boldsymbol{\alpha}}, \overline{\boldsymbol{\alpha}})}, \overline{(\underline{\boldsymbol{\alpha}}, \overline{\boldsymbol{\alpha}})}\Big)},\ \overline{\Big(\underline{(\underline{\boldsymbol{\alpha}}, \overline{\boldsymbol{\alpha}})}, \overline{(\underline{\boldsymbol{\alpha}}, \overline{\boldsymbol{\alpha}})}\Big)} \quad \text{delivers}$$
$$\underline{\Big(\Big(\underline{(\underline{\boldsymbol{\alpha}}, \overline{\boldsymbol{\alpha}})}, \overline{(\underline{\boldsymbol{\alpha}}, \overline{\boldsymbol{\alpha}})}\Big), \Big(\underline{(\underline{\boldsymbol{\alpha}}, \overline{\boldsymbol{\alpha}})}, \overline{(\underline{\boldsymbol{\alpha}}, \overline{\boldsymbol{\alpha}})}\Big)\Big)}; \ \ldots$$

This sort of informon-entropon recursion represents the holistic informational propagation of α-meaning through the informonic-entroponic consciousness system, being defined by the next assumption.

Assumption 5 Consciousness system *is an informonic-entroponic organization, denoted complexly by* $\Phi\big\lceil(\underline{\mathfrak{z}}, \overline{\mathfrak{z}})\big\rceil$. *Such a system connects in an informational way, through common operands, the informonic-entroponic spaces* $(\underline{\mathfrak{z}}, \overline{\mathfrak{z}})$ *and* $(\underline{\boldsymbol{\alpha_i}}, \overline{\boldsymbol{\alpha_i}})$ *in the form*

$$\Phi\big\lceil(\underline{\mathfrak{z}}, \overline{\mathfrak{z}})\big\rceil \rightleftharpoons \begin{pmatrix} (\underline{\mathfrak{z}}, \overline{\mathfrak{z}})\,; \\ (\underline{\boldsymbol{\alpha_i}}, \overline{\boldsymbol{\alpha_i}})\,; \\ i = 1, 2, \ldots, n; \\ n < \infty \end{pmatrix}$$

Relation $n < \infty$ *means that* n *is potentially not limited and rises with the informational development of the system.* □

Spaces $(\underline{\mathfrak{z}}, \overline{\mathfrak{z}})$ and $(\underline{\boldsymbol{\alpha_i}}, \overline{\boldsymbol{\alpha_i}})$ are informationally connected through common operands. Since each informon $\underline{\boldsymbol{\alpha_i}}$ is a conscious entity, on the primitive formula level, $\alpha_i\lfloor\ldots, \mathfrak{z}, \ldots\rfloor$ holds. On the informonic level, there is, certainly, $\underline{\boldsymbol{\alpha_i}}\lfloor\ldots, \underline{\mathfrak{z}}, \ldots\rfloor$ and, consequently, $\underline{\underline{\boldsymbol{\alpha_i}}\lfloor\ldots, \underline{\mathfrak{z}}, \ldots\rfloor}$.

3 Meaning of informational operands

One of the central notions of the informational and the conscious is the *meaning of informational entities.* One sort of understanding meaning comes from the use of natural language. Meaning is nothing more than the use of language within a community. In this way, meaning is the expression in language as such. The expression means that something comes to word where sequences of words appear.

Assumption 6 Meaning *concerns operands occurring in a formula, formula scheme and formula graph, and formula system, system scheme and system graph. Meaning of an operand is determined by the informational context with other operands and operators in different informational expressions. Meaning is nothing else than the expression of an operand, in a contextual or explicit way.* □

In a formula $\varphi^{\triangledown}_{\triangleright}\Big\lfloor\alpha_1, \alpha_2, \ldots, \alpha_i, \ldots, \alpha_{n_{\varphi^{\triangledown}_{\triangleright}}}\Big\rfloor$, where $\triangleright \in \{\rightarrow, \leftarrow, \rightleftarrows, (\rightarrow, \leftarrow)\}$ and $\triangledown \in \{\lambda, \circlearrowleft\}$, operand α_i is expressed contextually, within the formula between other operands. The scheme of this formula for $\triangleright = \rightarrow$ is

$$\mathfrak{S}\Big\lceil\varphi^{\triangledown}_{\rightarrow}\Big\lfloor\alpha_1, \alpha_2, \ldots, \alpha_i, \ldots, \alpha_{n_{\varphi^{\triangledown}_{\rightarrow}}}\Big\rfloor\Big\rceil \rightleftharpoons \Big(\alpha_1 \models \alpha_2 \models \ldots \models \alpha_i \models \ldots \models \alpha_{n_{\varphi^{\triangledown}_{\rightarrow}}}\,[\models \alpha_1]\Big)$$

From this scheme we see how α_i occurs in the context of other operands. Within this context, it has a specific meaning, so the conclusion can be made, what does it in fact mean. Option $[\models \alpha_1]$ in the scheme represents the circular case, where $\triangledown = \circlearrowleft$. In a circular situation, operand α_i in the scheme can be rotated to the initial position of the scheme, that is,

$$\mathfrak{S}\lceil\mu\lceil\alpha_i\rceil\rceil \rightleftharpoons \Big(\alpha_i \models \ldots \models \alpha_{n_{\varphi^{\circlearrowleft}_{\rightarrow}}} \models \alpha_1 \models \alpha_2 \models \ldots \models \alpha_i\Big)$$

where $\mu \rightleftharpoons \mathfrak{m}_{\text{meaning}}$. This kind of the scheme can be grasped as an explicit circular definition, that is, the one among parallel meaning schemes pertaining to operand α_i. The unambiguous meaning $\mu\lceil\alpha_i\rceil$ can be obtained by parenthesizing this scheme, for instance, by formula

$$\mu\lceil\alpha_i\rceil \rightleftharpoons \overset{\circlearrowleft}{\varphi}_{1\rightarrow}\Big\lfloor\alpha_i,\ldots,\alpha_{n_{\overset{\circlearrowleft}{\varphi}_{\rightarrow}}},\alpha_1,\alpha_2,\ldots,\alpha_i\Big\rfloor$$

Formally, the schematic result obtained by the operand rotation procedure can be expressed directly, using the rotation operand $\mathfrak{R}$ concerning operand α_i, in the form

$$\mathfrak{S}\lceil\mu\lceil\alpha_i\rceil\rceil \rightleftharpoons \mathfrak{R}\lceil\alpha_i\rceil\Big\lceil\mathfrak{S}\Big\lceil\overset{\circlearrowleft}{\varphi}_{\rightarrow}\Big\lfloor\alpha_1,\alpha_2,\ldots,\alpha_i,\ldots,\alpha_{n_{\overset{\circlearrowleft}{\varphi}_{\rightarrow}}}\Big\rfloor\Big\rceil\Big\rceil$$

Meaning $\mu\lceil\alpha_i\rceil$ is then a concrete parenthesizing of the scheme, that is,

$$\mu\lceil\alpha_i\rceil \rightleftharpoons \mathfrak{P}\Big\lceil\mathfrak{R}\lceil\alpha_i\rceil\Big\lceil\mathfrak{S}\Big\lceil\overset{\circlearrowleft}{\varphi}_{\rightarrow}\Big\lfloor\alpha_1,\alpha_2,\ldots,\alpha_i,\ldots,\alpha_{n_{\overset{\circlearrowleft}{\varphi}_{\rightarrow}}}\Big\rfloor\Big\rceil\Big\rceil\Big\rceil$$

where $\mathfrak{P}$ is the operator of parenthesizing.

A direct, serial, or linear meaning of an informational operand α_1 is given by a serial, non-circular formula in the form $\varphi_{\triangleright}\lfloor\alpha_1,\alpha_2,\ldots,\alpha_i,\ldots,\alpha_{n_{\varphi\rightarrow}}\rfloor$ putting

$$\alpha_1 \rightleftharpoons \varphi_{\triangleright}\lfloor\alpha_1,\alpha_2,\ldots,\alpha_i,\ldots,\alpha_{n_{\varphi_{\triangleright}}}\rfloor$$

For getting the meaning of formula system Φ's operand α_{ij_i}, it would suffice to put at the relation

$$\overset{\triangledown_i}{\varphi}_{i\triangleright_i}\Big\lfloor\alpha_{ij_i},\ldots,\alpha_{in_{\overset{\triangledown_i}{\varphi}_{i\triangleright_i}}},\alpha_{i1},\alpha_{i2},\ldots,\alpha_{i-1}\Big\rfloor \in \Phi,$$

$\alpha_{ij_i} \rightleftharpoons \Phi$. It is understood that system formulas

$$\overset{\triangledown_1}{\varphi}_{1\triangleright_1},\overset{\triangledown_2}{\varphi}_{2\triangleright_2},\ldots,\overset{\triangledown_i}{\varphi}_{i\triangleright_i},\ldots,\overset{\triangledown_{n_\Phi}}{\varphi}_{n_\Phi\triangleright_{n_\Phi}} \in \Phi$$

are mutually informationally connected (dependent, impacted) through common operands. In this view the meaning of α_{ij_i} becomes as complex as possible, expressed by system formulas, interpreting its meaning in different possible, explicit and implicit ways.

4 How to make an informational entity conscious?

By an informational entity the meaning is understood belonging to the explicitly expressed entity's title, its name. For instance, entity α has the name α and simultaneously represents α's meaning in the form of an informational formula $\alpha \rightleftharpoons \alpha\lfloor\alpha_1,\alpha_2,\ldots,\alpha_i,\ldots,\alpha_{n_\alpha}\rfloor$, schematized by $\mathfrak{S}\lceil\alpha\rceil \rightleftharpoons (\alpha_1 \models \alpha_2 \models \ldots \models \alpha_i \models \alpha_{n_\alpha})$, where some operands α_i may equal to α. A complex meaning of α is expressed by a formula system which could mean an explicit definition of operand α in the form

$$\alpha \rightleftharpoons \begin{pmatrix} \alpha\lfloor\alpha_{11},\alpha_{12},\ldots,\alpha_{1i_1},\ldots,\alpha_{1n_1}\rfloor; \\ \alpha\lfloor\alpha_{21},\alpha_{22},\ldots,\alpha_{2i_2},\ldots,\alpha_{2n_2}\rfloor; \\ \vdots \\ \alpha\lfloor\alpha_{j1},\alpha_{j2},\ldots,\alpha_{ji_j},\ldots,\alpha_{jn_j}\rfloor; \\ \vdots \\ \alpha\lfloor\alpha_{m1},\alpha_{m2},\ldots,\alpha_{mi_m},\ldots,\alpha_{mn_m}\rfloor \end{pmatrix}$$

schematized by

$$\mathfrak{S}\lceil\alpha\rceil \rightleftharpoons \begin{pmatrix} \alpha_{11} \models \alpha_{12} \models \ldots \models \alpha_{1i_1} \models \ldots \models \alpha_{1n_1}; \\ \alpha_{21} \models \alpha_{22} \models \ldots \models \alpha_{2i_2} \models \ldots \models \alpha_{2n_2}; \\ \vdots \\ \alpha_{j1} \models \alpha_{j2} \models \ldots \models \alpha_{ji_j} \models \ldots \models \alpha_{jn_j}; \\ \vdots \\ \alpha_{m1} \models \alpha_{m2} \models \ldots \models \alpha_{mi_m} \models \ldots \models \alpha_{mn_m} \end{pmatrix}$$

where some operands α_{ji_j} may equal α. System formulas $\alpha\lfloor\alpha_{j1},\alpha_{j2},\ldots,\alpha_{ji_j},\ldots,\alpha_{jn_{j\alpha}}\rfloor$ of the meaning system α can be mutually connected also by other common operands than α, where $j = 1,2,\ldots,m$ and $i_j = 1,2,\ldots,n_j$.

Assumption 7 *An informational entity, named by α, becomes* primitively conscious *if it is informationally connected with at least one conscious space $\left(\underline{\boldsymbol{\beta}},\overline{\boldsymbol{\beta}}\right)$ within a consciousness system $\Phi\left\lceil\left(\underline{\mathfrak{z}},\overline{\mathfrak{z}}\right)\right\rceil$. The ability of consciousness is granted to α transitively, via the conscious organization of informon $\boldsymbol{\beta}$. Thus, entity α transits into informon $\underline{\boldsymbol{\alpha}}$ and $(\underline{\boldsymbol{\alpha}},\overline{\boldsymbol{\alpha}}) \in \Phi\left\lceil\left(\underline{\mathfrak{z}},\overline{\mathfrak{z}}\right)\right\rceil$.* □

This kind of α's conscious emergence does not need $\underline{\boldsymbol{\alpha}}$'s own metaphysicalistic organization. So, $\underline{\boldsymbol{\alpha}}$ cannot decide autonomously upon its own conscious development and is, in this respect, dependent on informational spaces of other system operands.

Assumption 8 *An informational entity, named by α, becomes* properly conscious *if it is organized in the sense of initial informational metaphysicalism $\mathfrak{M}_{\triangleright}^{\circlearrowleft\parallel}\lceil\alpha\rceil$ and, then, complexly developed to the informonic-entroponic space $(\underline{\boldsymbol{\alpha}},\overline{\boldsymbol{\alpha}})$. Through metaphysicalism, informon $\underline{\boldsymbol{\alpha}}$ appropriates its own ability of consciousness, and the corresponding ability of subconsciousness, expressed entroponically by $\overline{\boldsymbol{\alpha}}$. This means that space $(\underline{\boldsymbol{\alpha}},\overline{\boldsymbol{\alpha}})$ is complexly connected to spaces $\left(\underline{\mathfrak{z}},\overline{\mathfrak{z}}\right)$ and $\left(\underline{\boldsymbol{\alpha_i}},\overline{\boldsymbol{\alpha_i}}\right)$, constituting the common consciousness system $\Phi\left\lceil\left(\underline{\mathfrak{z}},\overline{\mathfrak{z}}\right)\right\rceil$, where $(\underline{\boldsymbol{\alpha}},\overline{\boldsymbol{\alpha}}),\left(\underline{\mathfrak{z}},\overline{\mathfrak{z}}\right),\left(\underline{\boldsymbol{\alpha_i}},\overline{\boldsymbol{\alpha_i}}\right) \in \Phi\left\lceil\left(\underline{\mathfrak{z}},\overline{\mathfrak{z}}\right)\right\rceil$. Consciousness space $\left(\underline{\mathfrak{z}},\overline{\mathfrak{z}}\right)$ is meant to be the central informonic-entroponic space for the metaphysicalistically organized informational entity $\mathfrak{z} \rightleftharpoons \mathfrak{c}_{\text{consciousness}}$.* □

Informonic solution for operand α becomes, according to the previous formula system, as complex as

$$\alpha \rightleftharpoons \begin{pmatrix} \underline{\boldsymbol{\alpha}}\left\lfloor \underline{\boldsymbol{\alpha_{11}}}, \underline{\boldsymbol{\alpha_{12}}}, \ldots, \underline{\boldsymbol{\alpha_{1i_1}}}, \ldots, \underline{\boldsymbol{\alpha_{1n_1}}} \right\rfloor ; \\ \underline{\boldsymbol{\alpha}}\left\lfloor \underline{\boldsymbol{\alpha_{21}}}, \underline{\boldsymbol{\alpha_{22}}}, \ldots, \underline{\boldsymbol{\alpha_{2i_2}}}, \ldots, \underline{\boldsymbol{\alpha_{2n_2}}} \right\rfloor ; \\ \vdots \\ \underline{\boldsymbol{\alpha}}\left\lfloor \underline{\boldsymbol{\alpha_{j1}}}, \underline{\boldsymbol{\alpha_{j2}}}, \ldots, \underline{\boldsymbol{\alpha_{ji_j}}}, \ldots, \underline{\boldsymbol{\alpha_{jn_j}}} \right\rfloor ; \\ \vdots \\ \underline{\boldsymbol{\alpha}}\left\lfloor \underline{\boldsymbol{\alpha_{m1}}}, \underline{\boldsymbol{\alpha_{m2}}}, \ldots, \underline{\boldsymbol{\alpha_{mi_m}}}, \ldots, \underline{\boldsymbol{\alpha_{mn_m}}} \right\rfloor \end{pmatrix}$$

5 Informational organization of a language thesaurus

An exhaustive and adequately structured language thesaurus is the key means at the design and implementation of informational consciousness. It is a must for the design of *metaphysicalistically conceptualized artificial informational consciousness system* (MCAICS) being the major component of other possible concepts, theories, and implementations of consciousness systems.

In Fig. 1, the recursive graph of a more or less complete thesaurus of a natural language is presented [1]. This graph can be additionally meaningly refined according to the design needs, and shows, at the first glance, a linear or serial, tree-like structure. However, the graph becomes circular as soon as, in a concrete case, the word (informational operand) appears, being used in a previous (higher) knot of the graph. Such a circular situation is regular and can concern any headword in the dictionary.

Let us look a case from thesaurus [1] in which the headword *attention* is presented in the following way[1]:

attention *n* **1** a focusing of the mind on something <gave the problem careful *attention*>
syn application, concentration, consideration, debate, deliberation, heed, study
rel assiduity, diligence, industry, sedulity, sedulousness; notice, observation, regard, remark; absorption, engrossment, immersion, intentness
con absence, absentmindedness, abstraction, detachment, remoteness, withdrawal; disinterest, indifference, unconcern, unmindfulness
ant inattention
2 syn see NOTICE 1
rel awareness, consciousness, mindfulness, sensibility
con disregard, heedlessness, insensibility, unawareness, unconsciousness
3 syn see COURTESY 1
rel deference, homage, honor, reverence, benignity; considerateness, consideration, kindliness, solicitude
con neglect, negligence; aloofness, indifference, unconcern; discourtesy

notice *n* **1** a noting of or concerning oneself with something <take *notice* of the gathering clouds>
syn attention, cognizance, ear, head, mark ‖ mind, note, observance, observation, regard, remark
rel care, concern, consideration, thought; apprehension, grasp, understanding
con disinterest, disregard, indifference, unconcern; carelessness, heedlessness, unmindfulness; insouciance, negligence, recklessness
2 syn see MEMORANDUM 2
3 syn see CRITICISM 1

Etc. One sees how by the use of thesaurus the meaning of the headword *attention* expands in a positive and negative sense. However, a thesaurus includes also other words used in the subscript language of informational operands and informational operators (verbs, adjectives, prepositions, etc.) Between two subscribed operands, an appropriately subscribed operator must be chosen, meaningly corresponding to the context in which operands and operators occur. Thus, verbs or verbal phrases can be searched in a thesaurus too.

6 Where does a thesaurus meet informational metaphysicalism?

A thesaurus must not be understood as a headword-synonym dictionary but rather a much more complex interpretation of the headword meaning also in a contrasted, idiomatic, and other possibilities of positively and negatively related word meaning, concerning the headword [1]. In such a dictionary, synonyms, antonyms, related words, contrasted words, idiomatic equivalents, and other meaningly relevant words and phrases to a headword can be searched. Then, in the second step of searching, again, all these categories of words and phrases can be searched to a found synonym, antonym, related word, contrasted word, idiomatic equivalent, and other meaningly relevant word and phrase. This technique of the initial headword identification within a broader meaning can expand as deep as necessary, offering the sufficient informational complexity of the headword interpretation by sentences, concerning the searched thesaural entities. To this pure thesaural identification of a headword, sentences can be constructed, explaining or defining the headword meaning. In this way, a complex meaning of the headword can emerge expressing what the headword represents informationally in a context and what it does not represent at all.

A thesaurally represented headword α can come close to the concept of metaphysicalistic decomposition $\mathfrak{M}_{\triangleright}^{\bigcirc\|}$ concerning the headword α, that is, $\mathfrak{M}_{\triangleright}^{\bigcirc\|}\lceil\alpha\rceil$. Informational

[1] The meaning of abbreviations, also in further examples, is the following:
syn synonym(s) **rel** related word(s)
ant antonym(s) **con** contrasted word(s)
idiom idiomatic equivalent(s)
‖ use limited; if in doubt, see a dictionary

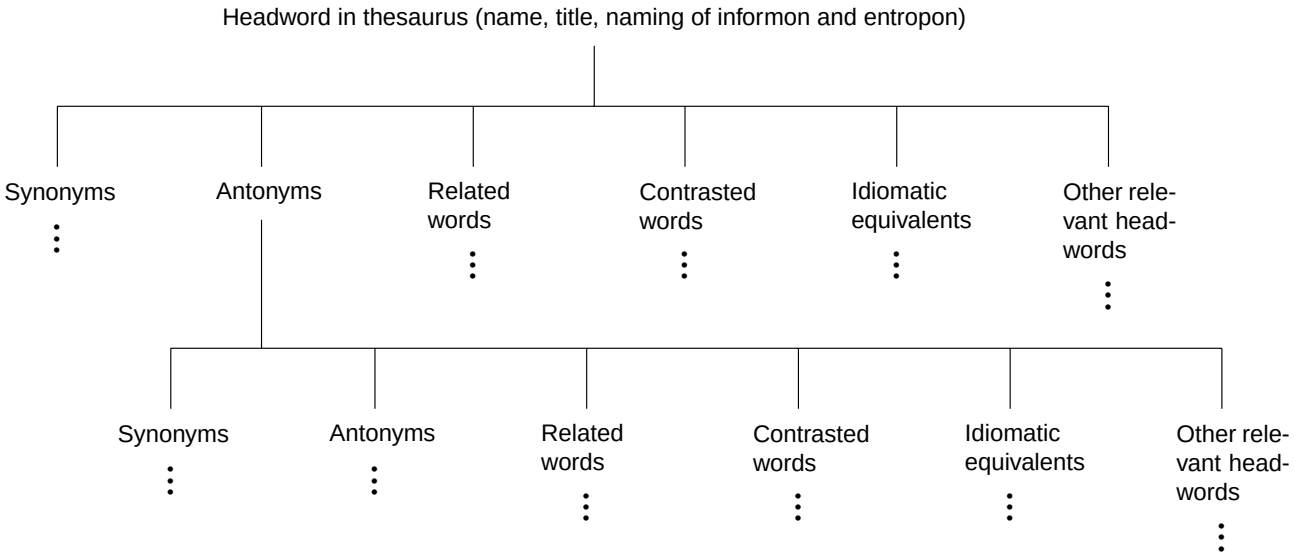


Figure 1: *Recursive organization of thesaurus [1] for the design of thesaural-metaphysicalistic structure developing informon. Such a thesaurus is a must in conceptualizing and design of informational consciousness.*

metaphysicalism is organized as informing, counterinforming, and informational embedding concerning an informational entity, in this case, the headword. Thesaural identification of a headword is structured

- by its synonyms and meaningly related words — a kind of positive identification in the sense of metaphysicalistic informing,
- by its antonyms and meaningly contrasted words — a kind of negative identification in the sense of metaphysicalistic counterinforming, and
- by a kind of meaningly idiomatic identification (an equalization) — in the sense of metaphysicalistic informational embedding (fixing the meaning).

Informational embedding can be grasped also as a kind of decision making upon the meaning, resulting from metaphysicalistic informing and counterinforming of the headword. To the metaphysicalistic-thesaural model belongs also the determination of concrete operators figuring in the metaphysicalistic organization which subscripts can be chosen by means of the thesaurus. The operand-operator identification in a metaphysicalistic model can be efficiently meaningly covered by words and word phrases obtained by the use of a thesaurus. Thesaural approach is in fact a complex approach to the problem of meaning concerning a headword and its identification up to the possible details of meaning or interpretation.

For each informational entity intending to inform in a conscious way, it is necessary to know what consciousness does mean in the most possible complex way of meaning expression. Therefore, the meaning of the concept "consciousness" must be developed in an informationally complex manner. What is needed is the informon $\mathfrak{c}_{\underline{\text{consciousness}}}$, represented by its informational synonym $\underline{\mathfrak{z}}$, and determined by the possible, informationally most complex form, from the thesaural point of view. Further, within the metaphysicalistic model of an operand α, one has to distinguish meaningly relatively closely positioned operands, like informing $\mathfrak{I}_\alpha$ and to it corresponding informational entity $\mathfrak{i}_\alpha$, counterinforming $\mathfrak{C}_\alpha$ and to it corresponding counterinformational entity $\mathfrak{c}_\alpha$, and informational embedding $\mathfrak{E}_\alpha$ and to it corresponding informational embedding entity $\mathfrak{e}_\alpha$. Informational embedding is a kind of decision making in the understanding of that which results from metaphysicalistic informing and the corresponding counterinforming within systemically organized entity's metaphysicalism.

Let us show an experiment made on the basis of Tab. 1, where metaphysicalistic components are listed both in the sense of thesaural contents and metaphysicalistic meaning (semantics). The table proceeds from name α, being the informational identifier of metaphysicalistic decomposition in the left-most column. The following columns represent metaphysicalistic informing $\mathfrak{I}_\alpha$ (second column), metaphysicalistic informational entity $\mathfrak{i}_\alpha$ (third column), metaphysicalistic counterinforming $\mathfrak{C}_\alpha$ (fourth column), metaphysicalistic counterinformational entity $\mathfrak{c}_\alpha$ (fifth column), metaphysicalistic informational embedding $\mathfrak{E}_\alpha$ (sixth column), and metaphysicalistic informational embedding entity $\mathfrak{e}_\alpha$ (seventh column). All of the columns include metaphysicalistically appropriate, thesaural, and intentional contents, dedicated for the future development of metaphysicalistic organization, up to the complex informonic and entroponic perplexity of occurring operands. Thus, the table becomes a significant reminder for the metaphysicalistic decomposition of the named operand α.

In the upper part of the table, the first column deals with the named operand, being for instance a noun or a noun phrase in a thesaurus, dictionary, or encyclopedia. The named operand will appear as a headword, idiom, or otherwise related word in thesaurus. After a time of de-

Name α; Informational identifier	Metaphysicalistic informing $\mathfrak{I}_\alpha$	Metaphysicalistic informational entity $\mathfrak{i}_\alpha$	Metaphysicalistic counterinforming $\mathfrak{C}_\alpha$	Metaphysicalistic counterinformational entity $\mathfrak{c}_\alpha$	Metaphysicalistic informational embedding $\mathfrak{E}_\alpha$	Metaphysicalistic informational embedding entity $\mathfrak{e}_\alpha$
Noun; Noun phrase	α-intentional informing	α-intentional entity	α-counterintentional informing	α-counterintentional entity	Informing of α-intentional embedding	α-intentional embedding entity
Noun headword in a thesaurus; Idiomatically named entity; Associatively, otherwise informationally related headwords in a thesaurus; α-named informon; …	Participles corresponding to name synonyms, name related headwords, name-idiomatic meaning; Synonymizing name-related concepts; α-intentional informonic and entroponic informing; …	Noun synonyms; Related nouns; Synonyms of noun synonyms; Idioms in a thesaurus; Intentionally name-related words and concepts; α-intentional informons' and entropons' environment; …	Participles corresponding to α- and $\mathfrak{i}_\alpha$-antonyms, synonyms of antonyms, and antonyms of synonyms, to related headwords and idioms concerning them; α-counterintentional entroponic and informonic informing; …	Antonyms; Contrasted words; Synonyms of antonyms; Idiomatic opposition; Counterintentionally name-related words and concepts; α-counterintentional informons' and entropons' environment; …	Informing of understanding, to it correlated headwords as participles; Generating meaning corresponding to name α, $\mu[\alpha]$, its intention $\mathfrak{i}_\alpha, \mu[\mathfrak{i}_\alpha]$, and its counterintention $\mathfrak{c}_\alpha, \mu[\mathfrak{c}_\alpha]$; α-intentional meaning, embedding informonic informing; …	Understanding of synonyms, antonyms, synonyms of synonyms and antonyms, antonyms of synonyms and antonyms, and entities related to them, in different possible ways, and recursively to arbitrary depths; Resulting informonic meaning of the α-named entity; …
Consciousness, $\mathfrak{z} \rightleftharpoons \mathfrak{c}_{\text{consciousness}}$; Informon $\underline{\mathfrak{z}}$	Consciousness intending $\mathfrak{I}_\mathfrak{z}$; Intending to be conscious, $\underline{\mathfrak{I}_\mathfrak{z}}$	Consciousness intention $\mathfrak{i}_\mathfrak{z}$; Intention to be conscious, $\underline{\mathfrak{i}_\mathfrak{z}}$	Consciousness counterintending; Counterintending to be conscious, $\underline{\mathfrak{C}_\mathfrak{z}}$	Consciousness counterintention; Counterintention to be conscious, $\underline{\mathfrak{c}_\mathfrak{z}}$	Consciousness intentional embedding, constitutionalizing to be conscious, $\underline{\mathfrak{E}_\mathfrak{z}}$	Consciousness intentional embedment, constituting meaningly to be conscious, $\underline{\mathfrak{e}_\mathfrak{z}}$
Awareness; Self-consciousness; Mind; Intellect; Intelligence; Mentality; Affects; Disposition; Temper; Opinion; Personality; Spirit; Reason; Atmosphere; Intuition; …	Being conscious; Being aware, awake, cognizant, sensible, alive; Perceiving; Cognizing; Apprehending; Rationalizing; Observing; Controlling thought; Marking by thought, will, mind, design; Conceiving; Innovating; …	Consciousness; Awareness, awakeness, cognizance, sensibility; Perception; Cognition; Apprehension; Rationalism; Observation; Control of thought; The thought-, will-, mind-. design-marked: Concepts; Innovation; …	Being subconscious, unaware, ignorant, insensible, unalive; Misperceiving; Dogmatizing; Misapprehending; Disregarding; Ignoring; Diversifying thought; Making obscure by thought, will, mind, design; Misconceiving; Scholasticizing; …	Subconsciousness, unawareness, insensibility, unawakenedness; Misperception; Dogmatism; Misapprehension; Disregard; Ignorance; Diversity of thought; The thought-, will-, mind-. design-obscured: Misconception; Scholasticism; …	Conscious embedding; Cognizing; Emotionalizing; Attending; Being intelligent; Motivating; Being homeostatic; Behaving; Recognizing; Observing; Concentrating; Examining; Inferring; Comprehending; Conceptualizing; Designing; …	Conscious embedment; Cognition; Emotions; Attention; Intelligence; Motivation: Homeostasis; Behavior; Recognition; Observation; Concentration; Examination Inference; Comprehension; Concepts; Design; …

Table 1: *A rough thesaural expansion of metaphysicalistic operand α, in the upper part of the table, interpreted by an example of consciousness $\mathfrak{z}$, in the lower part of the table.*

velopment by decomposition, the initially named operand α will result into informon $\underline{\boldsymbol{\alpha}}$, becoming a conscious entity through its complexity and informational perplexedness with other conscious entities, that is, through informons of a conscious system.

In the next six columns (2–7), the α-intentional and α-counterintentional (positively and negatively α-related) subjects come into the decompositional foreground. They mean that informing, counterinforming, and informational embedding of entities must follow the α-intentional and α-counterintentional platform, respectively. That concerns a definition process of the α-named operand, that is, what it is or what it represent, and what it is not or what it does not represent.

In the second column of the upper table part, metaphysicalistic informing $\mathfrak{I}_\alpha$ as an α-intentional component is treated. It is expressed as participles corresponding to α-synonyms, α-related headwords, α-idioms meanings in a thesaurus, for instance. In this way, α-related concepts can be synonymized. This kind of intentional informing results gradually into α-informonic and α-entroponic components, building up the informational space of α-informing.

Consequently, the third column of the upper table deals with metaphysicalistic informational entity $\mathfrak{i}_\alpha$, through which the most substantial α-intentional platform is emerging. This happens intentionally by the inclusion of α-synonyms and whichever α-related concepts into the organization of $\mathfrak{i}_\alpha$, that is, in entity's informational context. Gradually, such a gathering of α-intentional concepts leads to emerging of α-informon and α-entropon with α-specific intention.

Counterinforming in columns four and five is being characterized by an opposing informing to the α-intentional informing, searching for contraries and contrasts, also negations in regard to α-positive meaning. α-counterinforming brings in the foreground the possibility to say what does α-entity not mean, that is, its anti-definition. By this, the domain of meaning for an entity can be expressed in a negative way of α-understanding and α-interpretation. α-conterinforming is on the way to produce such an α-negative domain of meaning in the form of α-conterinformational entity $\mathfrak{c}_\alpha$ in column five.

In the fourth column of the upper part of the table, metaphysicalistic conterinforming $\mathfrak{C}_\alpha$ as an α-counterintending component comes into the scope of discussion as an α-counterintending entity. The informing of this specific component dictates the searching and including of participles corresponding to α- and $\mathfrak{i}_\alpha$-antonyms, α- and $\mathfrak{i}_\alpha$-synonyms of antonyms, and α- and $\mathfrak{i}_\alpha$-antonyms of synonyms to related headwords and idioms concerning them. Through complex counterinforming, α-counterinforming informons and entropons can come into informational existence, spreading the negative or opposing meaning of operand α.

The fifth column of the upper part of the table deals with conterinformational entity $\mathfrak{c}_\alpha$ as an α-counterintentional component emerging as a consequence of otherwise informationally involved entities of the system. Here, the α-counterintentional nature comes to the informational surface. α-antonyms, α-contrasted words, and synonyms of α-antonyms pervade in entity's organization. α-idiomatic opposition comes into interpretation of operand α within the formula system represented by $\mathfrak{c}_\alpha$. Counterintentionally α-related words, word phrases, sentences, paragraphs, and concepts come into the informational texture of $\mathfrak{c}_\alpha$. This kind of phenomenalism results in $\mathfrak{c}_\alpha$-informons and $\mathfrak{c}_\alpha$-entropons ($\underline{\mathfrak{c}_{\boldsymbol{\alpha}}}$, $\overline{\mathfrak{c}_{\boldsymbol{\alpha}}}$, and others) constituting the counterinformational space of α.

Informational embedding filters, modulates, and unites α-intentionally the meaning of α-informing and α-counterinforming and performs as a kind of understanding of the α-named operand in a complex informational environment. By informational embedding, metaphysicalistic components inform as characteristic, initially determined components of a conscious system, following the cognitive-emotional paradigm together with other possible conscious constituents. The embedding entity is an α-intentional summary expressing positively and negatively informational properties of name α, that is, its meaning.

In the sixth column of the upper part of the table, dealing with metaphysicalistic α-informational embedding $\mathfrak{E}_\alpha$, participles corresponding to the field of understanding and interpretation, and to them correlated headwords or phrases, are considered. Here, the summarized α-meaning $\mu\lceil\alpha\rceil$ is being generated, taken into account both the α-intentional and α-counterintentional meaning, $\mu\lceil\mathfrak{i}_\alpha\rceil$ and $\mu\lceil\mathfrak{c}_\alpha\rceil$, respectively. In the form of informing of embedding, the informonic α-intentional meaning is coming into existence together with accompanying entropons of involved entities.

In the last, seventh column of the upper part of the table, metaphysicalistic informational embedding entity $\mathfrak{e}_\alpha$ is confronted with the α-intentional embedment of components resulting during the informing of α-intentionally and α-counterintentionally oriented informational environment. Here, different forms (formulas, formula systems corresponding to sentences, paragraphs, etc., respectively) of terminal understanding of synonyms, antonyms, synonyms of synonyms and antonyms, antonyms of synonyms and antonyms, entities related to all of them come into informational existence, recursively (circularly) to arbitrary depth of meaning. The result is the generated α-informonic and α-entroponic meaning of the α-named entity in the left-most column of the table.

The lower, concrete part of Tab. 1, can be called the thesaurus constructed for the headword "consciousness", as needed in the design of initial and, later, informonic metaphysicalistic organization, $\underline{\mathfrak{z}}$, in Fig. 2. The first step can be made by the use of an existing thesaurus (for instance, [1]) or dictionary, where some word meaning can be found. More adequate or professional approach is considering the cognitive-emotional paradigm in the informational study and design of conscious entities, using concepts and cat-

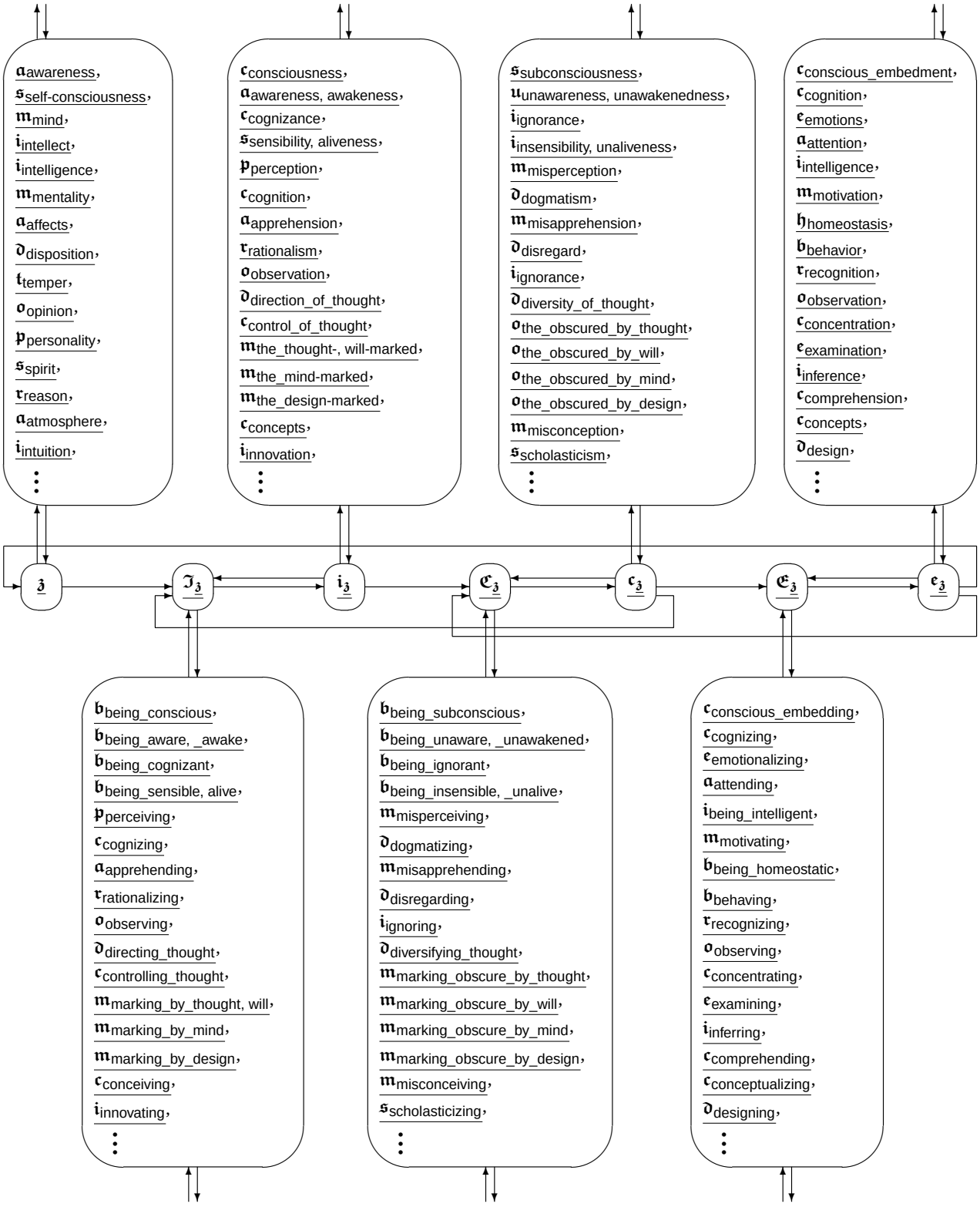


Figure 2: *Informational formalization of the lower part of Tab. 1, representing the graph of consciousness $\mathfrak{z}$ and its components in informonic form, showing how entity $\mathfrak{z}$ becomes conscious. As seen from the graph, initial metaphysicalistic decomposition $\mathfrak{M}^{\bigcirc\|}_{\rightarrow}\lceil\mathfrak{z}\rceil$ grows into the corresponding informonic complex.*

alogs of cognitive and emotional components, as shown elsewhere [2, 4, 7, 9].

Some of hints to the meaning of "consciousness" can be identified as

consciousness (in [6]) *n* **1 a :** the quality or state of being

aware esp. of something within oneself **b :** the state or fact of being conscious of an external object, state, or fact **c :** AWARENESS; *esp* **:** concern for some social or political case **2 :** the state of being characterized by sensation, emotion, volition, and thought **:** MIND **3 :** the totality of conscious states of an individual **4 :** the normal state of conscious life **5 :** the upper level of mental life of which the person is aware as contrasted with unconscious processes

aware (in [1]) *adj* marked by realization, perception, or knowledge often of something not generally realized, perceived, or known <*aware* of her own inner weakness>
syn alive, apprehensive, au courant, awake, cognizant, conscious, conversant, knowing, mindful, sensible, sentient, ware, witting
rel acquainted, appraised, informed; alert, heedful; impressionable, perceptive, receptive
con anesthetic, impassible, insensible, insensitive; ignorant, unknowing
ant unaware

conscious *adj* **1 syn** see AWARE
rel noticing, noting, observing, perceiving, remarking; vigilant, watchful
con forgetful, oblivious, unmindful; disregarding, ignoring, overlooking
ant unconscious
2 syn see SELF-CONSCIOUS

From nouns, adjectives, verbs, etc. in dictionaries and thesauri adequate operand names for metaphysicalistic operands can be chosen as shown in Tab. 1 and Fig. 2. Entering into recursive depth of dictionaries and thesauri, the complexity of operands and their mutual connection can rise up to the required degree and precision of concrete operand meanings.

The metaphysicalistic organization of operand $\mathfrak{z}$, that is, of $\mathfrak{c}_{\mathsf{consciousness}}$, in the graph of Fig. 2, is informonized, that is, complexly informationally interweaved within a conscious system $\Phi\lceil(\underline{\mathfrak{z}},\overline{\mathfrak{z}})\rceil$ discussed in Assumption 5. For the initial metaphysicalistic organization, the serial decomposition $\mathfrak{M}_{\rightarrow}^{\circlearrowright\|}\lceil\mathfrak{z}\rceil$ was taken and, then, informonically organized according to the operand suggestions in Tab. 1.

On the other hand, nothing was determined on the level of graph operators, which in the same way as operands can be determined by adequate subscripts fitting the operator position between concrete operands. Namely, the general operator $\models$, as an informational joker, can be in principle put between arbitrary operands. However, in a concrete situation, the operator is subscribed according to the used natural language, expressing a verb or a verbal phrase (in the last case by an operator composition, e.g., as the operator composition in $\alpha \models_\alpha \circ \models_\beta \beta$).

7 Methodology of the design of informational consciousness

In the preceding section several possible and essential methods of designing informational consciousness have been demonstrated. Systematically, the methods can be formulated as follows.

Method 1 [The naive expansion of consciousness.] *The most general, in fact, non-structural or naive way of making an informational entity α conscious is to include it in an existing conscious system $\Phi\lceil(\underline{\mathfrak{z}},\overline{\mathfrak{z}})\rceil$ informonically, that is, $\underline{\alpha} \in \Phi\lceil(\underline{\mathfrak{z}},\overline{\mathfrak{z}})\rceil$, connecting it to informonic operands of system $\Phi\lceil(\underline{\mathfrak{z}},\overline{\mathfrak{z}})\rceil$ through common operands.* □

This method needs additional explanation. Operand α, understood as a formula or formula system, is so far a rough informational entity determined as a usual informational formula (also as a single operand) or informational formula system. As such, it has not the property of being conscious, but is an unconscious entity yet. Such a case can happen in reading a sentence in a foreign language. Then, the unknown operands can be linked to the known translated operands and the similar is done with unknown operators. Thus, the sentence becomes linked to already conscious (common, that is, translated) operands in the context of meaningly known entities. The sentence becomes understandable, that is consciously perceivable.

The second method intends to give a future conscious entity a primordial organization for the development into a proper self-content conscious informational entity, proceeding from the initial, thesaurally-metaphysicalistically organized structure. This method puts in the foreground the beginning of the design of an informational consciousness system when at least some key components must be developed first to the level of conscious informational behavior. One of such entities is without doubt the complex meaning of the headword 'consciousness'.

Method 2 [The thesaural-metaphysicalistic expansion of consciousness.] A headword, $\mathfrak{h} \rightleftharpoons \mathfrak{h}_{\mathsf{headword}}$, of a complete informational thesaurus, denoted by $\mathfrak{t}_{\mathsf{thesaurus}}$, with $\mathfrak{h} \in \mathfrak{t}_{\mathsf{thesaurus}}$, is made conscious by its informonic expansion $\underline{\mathfrak{h}}$ through a consequent recursive use of $\mathfrak{t}_{\mathsf{thesaurus}}$ in a metaphysicalistic way, that is, proceeding from the initial decomposition $\mathfrak{M}_{\triangleright}^{\circlearrowright\|}\lceil\mathfrak{h}\rceil$ to the informonic organization $\underline{\mathfrak{M}_{\triangleright}^{\circlearrowright\|}\lceil\underline{\mathfrak{h}}\rceil}$, building up the informational space $(\underline{\mathfrak{h}},\overline{\mathfrak{h}}) \rightleftharpoons \left(\underline{\mathfrak{M}_{\triangleright}^{\circlearrowright\|}\lceil\underline{\mathfrak{h}}\rceil},\overline{\mathfrak{M}_{\triangleright}^{\circlearrowright\|}\lceil\underline{\mathfrak{h}}\rceil}\right)$. □

What would the entroponic decomposition of entropon $\overline{\mathfrak{h}}$, $\overline{\mathfrak{M}_{\triangleright}^{\circlearrowright\|}\left\lceil\overline{\mathfrak{h}}\right\rceil}$, and the informonic decomposition of entropon $\overline{\mathfrak{h}}$, $\underline{\mathfrak{M}_{\triangleright}^{\circlearrowright\|}\left\lceil\overline{\mathfrak{h}}\right\rceil}$, mean at all? As already shown in Sect. 2,

$$\left(\underline{\mathfrak{M}_{\triangleright}^{\circlearrowright\|}\lceil\underline{\mathfrak{h}}\rceil},\overline{\mathfrak{M}_{\triangleright}^{\circlearrowright\|}\lceil\underline{\mathfrak{h}}\rceil}\right) \quad \text{delivers}$$

$$\left(\underline{\left(\underline{\mathfrak{M}_{\triangleright}^{\circlearrowright\|}\lceil\underline{\mathfrak{h}}\rceil},\overline{\mathfrak{M}_{\triangleright}^{\circlearrowright\|}\lceil\underline{\mathfrak{h}}\rceil}\right)},\overline{\left(\underline{\mathfrak{M}_{\triangleright}^{\circlearrowright\|}\lceil\underline{\mathfrak{h}}\rceil},\overline{\mathfrak{M}_{\triangleright}^{\circlearrowright\|}\lceil\underline{\mathfrak{h}}\rceil}\right)}\right)$$

as recursive expansion, to which the pointed-out entroponic decomposition of entropon and informonic decomposition of entropon do not belong. In a general case of operand α,

– *entropon of entropon*, an up to now not discussed entity $\overline{\overline{\boldsymbol{\alpha}}}$ or, more precisely, $\overline{(\overline{\boldsymbol{\alpha}})}$, could mean entropon $\overline{\boldsymbol{\alpha}}$ developing (emerging, coming into unconsciousness, expanding it) in an entroponic way, and

– *informon of entropon* is an up to now not discussed entity $\overline{\underline{\boldsymbol{\alpha}}}$, understood ambiguously as both $\underline{(\overline{\boldsymbol{\alpha}})}$ and $\overline{(\underline{\boldsymbol{\alpha}})}$; $\underline{(\overline{\boldsymbol{\alpha}})}$ could mean entropon $\overline{\boldsymbol{\alpha}}$ developing (emerging, coming into consciousness in the very moment) informonically, and $\overline{(\underline{\boldsymbol{\alpha}})}$ could mean informon $\underline{\boldsymbol{\alpha}}$ entroponizing in the very moment.

Thus, decomposition $\underline{\mathfrak{M}_{\triangleright}^{\circlearrowleft\shortparallel}\left[\overline{\mathfrak{h}}\right]}$ means entropon $\underline{\mathfrak{h}}$, which fragments in this very moment come into consciousness in a metaphysicalistic way, and decomposition $\overline{\mathfrak{M}_{\triangleright}^{\circlearrowleft\shortparallel}\left[\overline{\mathfrak{h}}\right]}$, which fragments in this very moment emerge unconsciously, that is, entroponically, in a metaphysicalistic way.

Consequence 1 [Determination of operator subscripts by means of a thesaural-metaphysicalistic approach.] *By the choice of headwords or headword phrases, denoted by* $\mathfrak{h}_1$ *and* $\mathfrak{h}_2$*, where* $\mathfrak{h}_1, \mathfrak{h}_2 \in \mathfrak{t}_{\text{thesaurus}}$*, the operator subscript (verb or verb phrase) for* $\models_{\text{subscript}} \in \mathfrak{t}_{\text{thesaurus}}$*, in a basic or complex* (*interiorly parenthesized* $\mathfrak{h}_1$*- and* $\mathfrak{h}_2$*-transition* $\mathfrak{h}_1 \models_{\text{subscript}} \mathfrak{h}_2$)*, depends on meaningly context determined circumstances, where operator* $\models_{\text{subscript}}$ *must fit the operator composition* $\models_{\mathfrak{h}_1\text{-subscript\_dependent}} \circ \models_{\mathfrak{h}_2\text{-subscript\_dependent}}$*, that is,*

$$(\mathfrak{h}_1 \models_{\text{subscript}} \mathfrak{h}_2) \rightleftharpoons (\mathfrak{h}_1 \models_{\mathfrak{h}_1\text{-subscript\_dependent}} \circ \models_{\mathfrak{h}_2\text{-subscript\_dependent}} \mathfrak{h}_2)$$

Operator subscript dependence is a matter of meaning within the natural language context, that is, the use of language. □

In Fig. 2, all the paths of the graph, representing operators, are informational jokers $\models$, with the general meaning of the verb "to inform". They are not subscribed yet. For instance, the operator of informational concern $\models_{\Psi}$ could be used in many cases. However, more specific operators from the thesaurus, appearing in headword definitions, can be chosen, e.g., $\models_{\text{be\_conscious}}, \models_{\text{be\_aware}}, \models_{\text{be\_unconscious}}, \models_{\text{conceive}}, \models_{\text{perceive}}, \models_{\text{understand}}, \models_{\text{intend}}$, etc.

8 Conclusion

The design of a thesaurus belongs to the hard problems of linguistic study, being an innovative effort in the direction of a deeper understanding of language, of enlarging and spreading the word and phrase meaning.

It is important to comment where the informational approach to conscious agents could meet the Computing Research Repository (ACM) as an innovative research theory and practice on the way to informational consciousness. This paper shows how the subject concerns natural language processing [Computation and Language], combinatorics and graph theory [Discrete Mathematics], indexing (naming), dictionaries (thesauri), retrieval, content and analysis (meaning) [Information Retrieval], multiagent systems (informons), distributed artificial intelligence, intelligent agents, coordinated interactions, and practical applications [Multiagent Systems], connectionism and adaptive behavior [Neural and Evolutionary Computation], other not listed subject areas (consciousness) [Other], and robotics in the sense of informational consciousness [Robotics] [5]. Informational consciousness directly concerns cognitive psychology, cognitive science [2, 4], linguistics, and the philosophy of information [3] and mind.

This paper and [8] show how in English and German, respectively, it is possible to begin the design and programming of informational consciousness system by the use of language thesaurus. It becomes evident that particular thesauri have to be constructed concerning the consciousness related terms of a natural language. The process of translation could be hidden behind a bit more complex organization of informational conscious system: in the first phase, the first natural language is translated into the first (subscribed) informational language. In the second phase, the mapping of the first informational language into the second informational language takes place. This mapping can be determined in advance by approved rules between two languages. In the third phase, the obtained result in the second informational language is translated into the second natural language.

References

[1] THE MERRIAM-WEBSTER CONCISE SCHOOL AND OFFICE THESAURUS. 1991. Merriam-Webster Inc., Publishers. Springfield, MA.

[2] DALGLEISH, T. & M. POWER, EDS. 2000. Handbook of Cognition and Emotion. John Wiley & Sons. Chichester, England.

[3] FLORIDI, L., Ed. 2003/2004. The philosophy of information. Minds and Machines 13:459–588/14:1–132.

[4] LEWIS, M. & J.M. HAVILAND-JONES, Eds. 2000. Handbook of Emotions. Second Edition. The Guilford Press. New York, London.

[5] MORAVEC, H. 1999. Robot. Mere Machine to Transcendent Mind. Oxford University Press. New York.

[6] WEBSTER'S NINTH NEW COLLEGIATE DICTIONARY 1986. Merriam-Webster, Inc. Springfield, MA.

[7] ŽELEZNIKAR, A.P.[2] 2003. Conscious informational entities. Informatica 27:483–494.

[8] ŽELEZNIKAR, A.P. 2004. Informon und Entropon im Bewusstseinssystem. Grundlagenstudien aus Kybernetik und Geisteswissenschaft/Humankybernetik 45:81–89.

[9] ŽELEZNIKAR, A.P.[2] 2004. Introduction to Artificial Consciousness. The Philosophy of the Informational, Formalization, and Implementation. A study in progress.

[2]Readable also in PDF (Adobe Acrobat Reader), at the website <**http://www.artifico.org**>.

The Demarcate Construction: A New Form of Tree-based Priority Queues

Rick Siow Mong Goh, Wai Teng Tang, Ian Li- Jin Thng and Marie Therese Robles Quieta
National University of Singapore, Department of Electrical and Computer Engineering
3 Engineering Drive 3, CCN Laboratory, Singapore 117576
<engp1815,eng90464,eletlj,g0202825@nus.edu.sg>

Keywords: priority queue, splay tree, skew heap, calendar queue

Received: February 22, 2004

A new form of tree-based priority queues is proposed. These priority queues employ the demarcation process to systematically split a single tree-based priority queue into many smaller trees, each divided by a logical time boundary. These new Demarcate Construction priority queues offer an average speedup of more than twice over the single tree-based counterparts and outperform the current expected O(1) Calendar Queues in many scenarios. Their generality in small to large queue sizes, insensitivity to priority increment distributions and low overhead costs, render them suitable for many applications.

Povzetek: članek opisuje novo obliko prioritetnih vrst.

1 Introduction

In this article we apply the concepts of *demarcation* to tree-based priority queues. We define demarcation as the process where by events in a tree-based priority queue are separated clearly by time-boundaries in the form of *buckets*, where each bucket is made up of a single tree.

We are motivated by the observation that the known kinds of efficient trees such as the Splay Tree [1] and Skew Heap [2] only have at best an *amortized* time bound of $O(\log(n))$ per operation, where by amortized time is meant the time per operation averaged over a worst-case sequence of operations [3]. Comparatively, multilist-based priority queues such as the Calendar Queue (CQ) [4] and its variant Dynamic CQ (DCQ) [5] offer an "expected" $O(1)$ average time bound per operation, where by "expected" is meant that the CQs are not theoretically proven to be $O(1)$ but rather displays an $O(1)$ performance in numerous application scenarios. However, the drawback of employing the CQs in applications is that the worst-case time bound per operation can be as poor as $O(n)$ [6].

That said, the CQ has nonetheless been a popular choice as the pending event set structure in discrete event simulators such as CSIM18 [7], GTW [8], Network Simulator v2 [9], as well as in a quality of service algorithm where it maintains the real-time packet requests [10] and even as part of a rate controller for ATM switches [11]. Similarly, tree-based priority queue – the Splay Tree too has a wide array of applications: the pending event set structure in the CelKit (formerly known as SimKit) simulator [12], the data structure for fast IP lookups [13], data compression [14] and is also used in the block sorting process of Burrows and Wheeler [15].

An important use of priority queues, both tree and multilist-based, is in the area of discrete event simulation (DES). In DES, the pending event set (PES) is defined as the set of all events generated during a DES which have not been simulated yet. The PES is in essence a priority queue controlling the flow of the simulation of events with the minimum timestamp having the highest priority and maximum timestamp having the least priority.

Comfort [16] has revealed that up to 40% of the computational effort in a simulation may be devoted on the management of the PES alone, where the *enqueue* and *dequeue* operations account for as much as 98% of all operations on the PES. A DES frequently operates in a three-step cycle: *dequeue* – removal of an event with the highest priority from the PES; *execute* – processing this dequeued event; *enqueue* – insertion of new event/s resulting from the execution into the PES. The two basic operations, enqueue and dequeue, have run-time complexity closely dependent on the total number of events in the PES. Therefore, a PES structure should be efficient especially for large-scale simulations that involve large number of events during simulation jobs.

In most applications the metric of interest for a priority queue is often the time required to perform the most common operations. This metric is referred to as *access* time. In typical applications such as DES, the total run-time of the simulation job is by far more important than the individual times of the operations, except for real-time applications. Therefore, the amortized (or average) access time per operation is by far more important than the worst-case access time for each individual operation. Fine-grain simulations, such as but not limited to ATM network simulations, are time-consuming due to the huge number of events to process [5]. The faster and the larger

the networks, the higher the number of events would be in the PES and the longer run-times these network simulations would require, which may take days or weeks to yield results with an acceptable level of statistical error. For example, experiments conducted in Tcpsim [17] for a three-minute simulated time over Sun Ultra 1 took more than one day execution time on average [5]. Therefore, to speed up simulation jobs, one approach is to develop high-performance priority queue structures for the PES.

In this article we develop the *Demarcate Construction* (*Demarco*) priority queue, a multilist-based structure which is made up of two building blocks. The name *Demarco* arises from the word "demarcate" which means to divide and separate clearly as if by boundaries. The primary structure is an array of buckets, where each bucket may contain a tree holding *near-future* events. The secondary structure is made up of a simple unsorted linked list to hold *far-future* events. *Demarcation* refers to the process of constructing the primary structure and transferring events from the secondary structure to the primary. In an amortized sense, this demarcation process ensures that a tree-based priority queue has comparable performance or better, than one which does not undergo demarcation.

The rest of this article is organized as follows: Section 2 describes in detail the *Demarcate Construction*, Section 3 explains the performance measurement techniques and the empirical results from various experiments using these techniques are given in Section 4. Finally, Section 5 concludes.

2 Demarcate Construction

The *Demarcate Construction* (*Demarco*) has four essential principles.

First and foremost, the concept of demarcation is to have many trees each containing a small number of events. In contrast, a tree-based priority queue manages only a single tree containing all the nodes or events. Upon applying demarcation, an array of logical buckets is constructed. Each bucket spans equal time-interval and these buckets systematically enable the events to be demarcated and distributed in the buckets. Thus on the average, the tree in each bucket will have a smaller number of events leading to a much reduced height as compared to a single tree priority queue.

Secondly, *Demarco* defers the sorting of events until necessary. At the onset, all enqueued events are appended in the secondary tier (*SecT*) of *Demarco*. These events are not sorted according to their timestamps. During the first dequeue operation, the primary tier (*PriT*) is constructed and the events are inserted into the corresponding buckets in *PriT* where they are sorted according to the tree-based priority queue's native enqueue algorithm.

Thirdly, unlike other multilist-based priority queues, *Demarco* does not rely on sampling heuristics to obtain structure parameters. The parameters used when constructing *PriT* are obtained from the events distribution in the *SecT*.

Lastly, the algorithm of *Demarco* proceeds in demarcation *cycles* where by a *cycle* is defined as the duration when: the events in *SecT* are transferred to the *PriT*, more events are enqueued in *PriT* and *SecT*, and all the events in the *PriT* are dequeued.

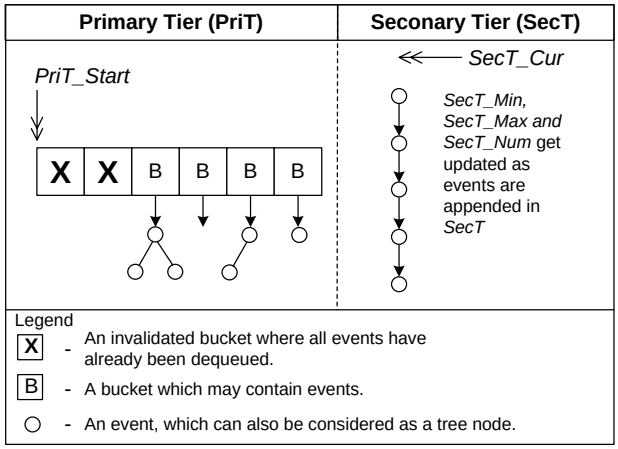


Figure 1: Basic structure of a *Demarcate Construction*.

2.1 Basic structure of *Demarco*

Figure 1 shows an example of the basic structure of a *Demarcate Construction* (*Demarco*). The main building blocks of the *Demarco* consist of:

1. Primary Tier (*PriT*) – an array of buckets where each bucket may contain a tree. Each tree-node contains an event holding a near-future (i.e. soon to be dequeued) timestamp. Within each bucket, the events are sorted according to the algorithm of the tree-based priority queue. The parameters used in creating the *PriT* are obtained from the events distribution in *SecT*. In Figure 1, there are a total of six buckets of which two of them are invalidated (i.e. events have already been dequeued before) and four buckets which may contain events.
2. Secondary Tier (*SecT*) – an unsorted singly linked list. Acting as an overflow list to contain far future events, *SecT* buffers events that do not affect the *PriT*. This reduces the number of events in the *PriT* and thus, on the average, the number of events in each bucket decreases as simulation time progresses. Since the performance of tree-based priority queues depends on the height (or number of levels), reducing the number of events in *PriT* will eventually lead to a reduction in the height of the tree in the buckets in *PriT*. This without doubt leads to a superior overall performance.

2.2 The *Demarco* algorithm

Demarco marks the first departure from the CQs' resize triggers and sampling heuristics to obtain structure

parameters such as the number of buckets and the bucketwidth. Instead of the static methodologies used in the CQs, *Demarco* employs a dynamic approach of updating its structure parameters, which is explained in Section 2.2.1.

The *Demarco* structure keeps a set of variables to function and they are defined as follow:
PriT_Start – Used for calculating the bucket-index of the event which is to be enqueued in *PriT*. It is set to *SecT_Min* during each Demarcation process, where by events are transferred from *SecT* to *PriT*.
PriT_Num – Number of events in *PriT*.
PriT_Bw – Bucketwidth of *PriT*.
PriT_Index – Bucket-index of the first non-empty bucket bucket in *PriT*.
SecT_Cur – Minimum timestamp of an event that can be enqueued in *SecT*. This value will be set equal to *SecT_Max* at each transfer of events from *SecT* to *PriT*.
SecT_Min – Minimum timestamp in *SecT*.
SecT_Max – Maximum timestamp in *SecT*.
SecT_Num – Number of events in *SecT*.

2.2.1 Dequeue operation

At the onset, all enqueued events are placed in *SecT* in a FIFO manner without time-order thus leaving *PriT* being empty. On the first dequeue operation, *PriT* is constructed and thereafter, all the events are transferred from *SecT* to *PriT*.

The bucketwidth of *PriT*, an important structure parameter, is dynamically assigned using equation (1).

$$PriT\_Bw = Bucketwidth = \frac{SecT\_Max - SecT\_Min}{SecT\_Num} \quad (1)$$

The number of buckets to be created in *PriT* is set to be *SecT_Num*, giving an average of one event per bucket on the assumption that the event distribution is a uniform distribution. Though in practical scenarios this may not be true, the *Demarco* will still perform well because the enqueue of events into *PriT* is $O(\log(n_B))$ per event whereby n_B is the number of events in a bucket. For most scenarios, $n_B << N$, where N is the total number of events in the *Demarco* structure.

After the construction of *PriT*, the events in *SecT* are transferred to *PriT*. Transferring of an event into *PriT* is alike enqueuing an event into *PriT* which utilizes the tree-based priority queue's native enqueue algorithm. Thereafter, the highest priority event would be in the first bucket in *PriT* (where *PriT_Index* = 0 and that *PriT*_Start = *SecT_Min* have been initialized). On each dequeue, the highest priority event would be removed from the first bucket in *PriT* by employing the tree-based priority queue's native dequeue algorithm.

Subsequently, when the first bucket is empty, it is *invalidated* and the second bucket is then considered, where at the same time, parameter *PriT*_Index is incremented by one.

If the second bucket is empty, *PriT_Index* is incremented again until a non-empty bucket is found and the current highest priority event is dequeued. After all the events in *PriT* are dequeued, i.e. all the buckets are empty, the demarcation cycle repeats itself with *SecT* treating the next dequeue to be alike the first dequeue as mentioned.

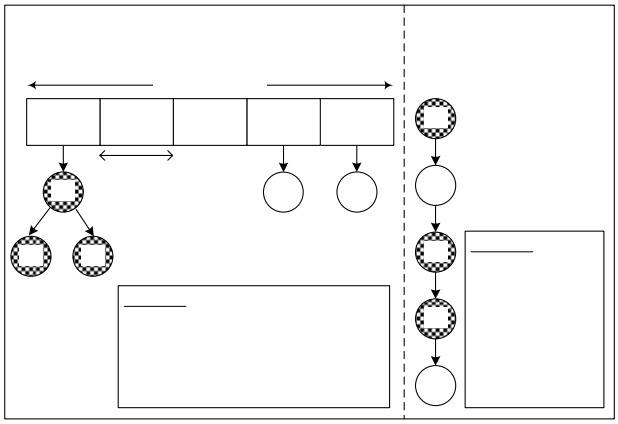


Figure 2: Dequeue operation.

Figure 2 illustrates an example of a demarcation process during a dequeue operation when there are events in *SecT* but *PriT* is empty. Essentially, *PriT* is created with parameters set according to the events found in *SecT*. Thereafter, events in *SecT* are transferred to *PriT*. Figure 2 demonstrates how the three smallest timestamp events (shaded) are being transferred from *SecT* to Bucket 0. For this example, we first assume that five events, with timestamps 0.5, 7.0, 1.1, 0.9 and 4.5, are enqueued in an empty *PriT*. Since *SecT_Cur* equals 0, all the events are enqueued in *SecT*. On the first dequeue, *PriT* is created with the parameters as stated in Figure 2 using equation (1). Thereafter, events from *SecT* are transferred to *PriT*. Immediately after this, the following variables are updated: *SecT_Cur* = *SecT_Max* = 7.0; *SecT_Min*, *SecT_Max*, *SecT_Num* are then reset. Events from the first non-empty bucket, i.e. Bucket 0, should now have already been sorted according to the tree's enqueue algorithm. Next, event 0.5 is dequeued, followed by 0.9 and 1.1 according to the tree's dequeue algorithm. After each dequeue *PriT_Num* is decremented and after dequeuing 1.1, *PriT_Num* should now be 2.

Subsequently, the algorithm searches for the next non-empty bucket, which is Bucket 3, and *PriT_Index* is updated as 3. Thereafter, event 4.5 is dequeued. When all the events in *PriT* are dequeued, the cycle repeats with *SecT* handling the next dequeue operation alike the first dequeue as mentioned earlier.

2.2.2 Enqueue operation

For each enqueue operation, *Demarco* checks if that event timestamp is greater than *SecT_Cur*. If so, the event is simply placed at the end of the linked list in *SecT*. If the event is not inserted in *SecT*, then the event is enqueued in *PriT*. On enqueuing in *PriT*, the bucket-

index of the bucket where this event is to be inserted in *PriT* is:

$$Bucket\_index = \frac{timestamp - PriT\_Start}{PriT\_Bw} \quad (2)$$

and the event is enqueued according to the tree's native enqueue algorithm.

2.3 Brief time complexity analysis

This section seeks to provide a brief performance analysis of a *Demarco* structure using the amortized time complexity analysis [3].

Assume N events are enqueued into an empty *Demarco* structure. Initially all the events would be enqueued into *SecT*. This enqueue of events into *SecT* is $O(1)$ per event since *SecT* is an unsorted linked list and the events are simply appended at the end of the linked list. On the first dequeue, all the events from *SecT* would be bucket-sorted into *PriT*, where *PriT* is made up of buckets with an equal time-interval (bucketwidth). Events would be distributed in *PriT* according to the event distribution. For instance, for a uniform distribution, each bucket would approximately hold one event.

Suppose during these dequeue operations, more events are enqueued into the *Demarco* structure. Since each bucket has equal time-interval, the events in *PriT* are likely to be spread out with each bucket having relatively small number of events. Therefore, even though a tree-based priority queue has $O(\log(n_B))$ complexity, n_B is expected to be $<< N$, leading to near $O(1)$ enqueue and dequeue complexity. Note that the cost of transferring the events from *SecT* to *PriT* is also $\sim O(N) / N = O(1)$ per event. Table 1 summarizes the theoretical performance of a *Demarco* structure with a tree-based priority which has $O(\log(n))$ amortized complexity for both its enqueue and dequeue operation.

Table 1: Amortized time complexity of a *Demarco* structure.

	PriT (expected, worst)	*SecT* (expected, worst)
Enqueue	$O(1)$, $O(\log(n))$	$O(1)$
Dequeue	$O(1)$, $O(\log(n))$	–

Through simulation benchmarks, we have shown empirically in Section 4 that a *Demarco* structure has near $O(1)$ performance under all distributions.

3 Performance Measurement Techniques

The performance of priority queues are often measured by the average access time to enqueue or dequeue an event under different load conditions. The parameters to be varied for each queue are: the access pattern model, the queue size and the priority distribution. The queue size refers to the number of events in a priority queue. For our experiments, the queue size ranges from 100 to 1 million events. This wide range represents what small to large-scale real simulation jobs will encounter.

3.1 Access pattern models

The access pattern models that have been proposed either emulate the steady-state or the transient phase of a typical simulation. They are as follow:

1. Classic Hold model [18]. The queue to be tested is initially built up to the specified benchmark size by using a random series of enqueues (with slightly higher probability) and dequeues. Thereafter a series of hold operations ensue. A hold operation is defined as a dequeue followed by an enqueue. The timestamp of a new enqueue event is obtained as follows: a random variable which describes the desired priority distribution to be tested is picked. The value of the random variable is then added to the timestamp of the event that was just dequeued and the result is the timestamp of the new enqueue event. The average access time to be calculated is the average time taken for one hold operation. The recommended number of hold operations, in order to obtain a reasonable accurate average, should be 30 times the queue size [6]. This method has the following features:
   a. The problem of determining the transient period is avoided, and
   b. The effect of the transient period on the different queue sizes tested are similar [6].
2. Up/Down model [19]. In an Up/Down test, the queue is built up to the benchmark queue size by a sequence of enqueues. Thereafter, the queue size is returned to zero by an equally long sequence of dequeues. The average access time to be calculated is the time taken for all queue operations (in the enqueue phase and dequeue phase), divided by the total number of queue operations. This Up/Down model emulates the worst case scenario of a simulation job which is not stable in queue size [19].

3.2 Priority increment distributions

The priority increment distributions, often used for the benchmarking of priority queue structures [5,6,18,19], are illustrated in Figure 3. The rand() used can be found in [20]. The Camel(x,y) distribution [19] represents a 2-hump heavily skewed distribution with x% of its mass concentrated in the two humps and the duration of the two humps is y% of the total interval. In addition to the five distributions as shown in Figure 3, the Change(A,B,x) distribution [19] was also used to test the sensitivity of the CQ when exposed to drastic changes in priority increment distribution. The compound distribution Change(A,B,x) interleaves two different priority increment distributions A and B together. Initially, x priority increments are drawn from A followed by another x priority increments drawn from B and so on. Change distributions can be used to model simulations where the priority increment distributions vary significantly over different time periods, for example

battlefield simulations. In our experiments, we use Camel(0,1000,0.001,0.999) and Change(Exp(1), Triangle(90000,100000),2000) which have been used in the landmark survey [6].

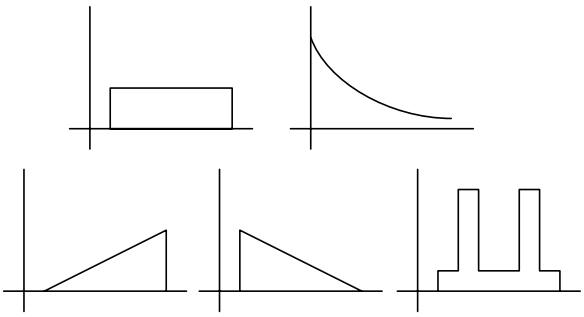


Figure 3: Priority increment distributions.

3.3 Benchmarking codes and hardware

The performance of the priority queues was obtained by conducting experiments on an AMD Athlon MP server. Even though this server has dual processors, the experiments did not make use of its true SMP capabilities as the algorithms are sequential. However, this server ensured that when each benchmark was carried out, other background processes that might affect the results were kept at a minimum, thus obtaining more accurate experimental results.

The code used for the Splay Tree was based on the Pascal code used by Jones [18]. Skew Heap implementation was based on the non-recursive code given by Jones [21]. SCQ and DCQ were based on the codes supplied by Brown [4] and Oh et al. [5] respectively. Two empirical tests were conducted to verify that no items in the priority queues were gained or lost and that successive dequeues removed events in stable time-order.

The experiments were performed with the required memory for each priority queue being pre-allocated. This was to eliminate the underlying memory management system which might affect the results. This is a good practice in actual DES as it prevents memory fragmentation when creating new events and deleting the serviced events. This method of pre-allocating memory would also enhance the performance of the DES. The method of pre-allocation could be made dynamic by an initial pre-allocation and subsequently, an allocation of memory on demand methodology could be employed.

All code was written in the C programming language with all recursive procedure calls and the like being eliminated. Loop overhead time and the time taken for random numbers generated were removed by factoring out the time required for running a dummy loop. Five runs of each experiment were done and the median value was obtained for each queue size simulated in the experiment. The median value is a measure of the central tendency and is chosen over the mean value because some background processes could have adversely affected a particular run of an experiment and averaging this value could render less accurate results.

4 Experimental Results

The objectives of this section are firstly to present the performance of tree-based priority queues with and without *Demarco*. Secondly, we compare *Demarco* priority queues with the current fastest multilist-based queues – CQ and DCQ. Lastly, we would like to determine *Demarco* priority queues' generality and sensitivity in the five priority increment distributions using the Classic Hold and Up/Down models, as well as when the queue size increases from 100 to 1 million. Note that a logarithmic scale has been used for the queue-size axis which leads to logarithmic complexity for linear plots.

4.1 Steady-state experiments

Figures 4(a) to 4(f) show the results obtained under the Classic Hold experiments which is commonly employed to test the steady-state performance of the priority queues. Note that the obvious knee seen in the graphs is due to the declining cache performance and occurs when the queue size is about 10,000. This phenomenon is also observed in the graphs in [6] where the experiments were done on SUN and Intel architectures.

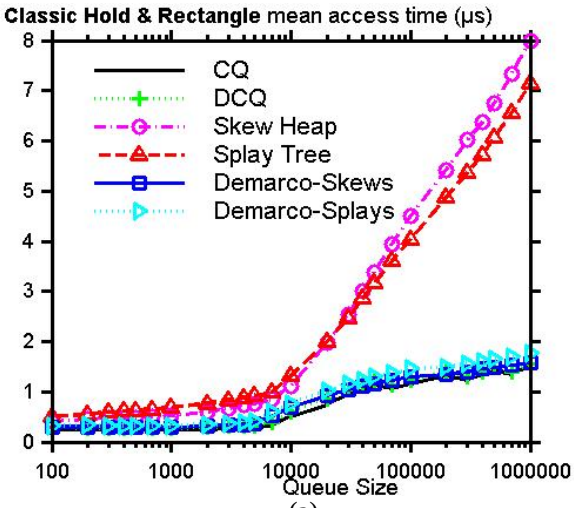
Figures 4 show vividly that the performance of *Demarco* structures, i.e. *Demarco*-Skews and *Demarco*-Splays, outperform the tree-based priority queues; Skew Heap and Splay Tree, where by *Demarco*-Skews/-Splays is made up of a *Demarco* structure where each bucket in *PriT* of *Demarco* may contain a Skew Heap/Splay Tree. At larger queue sizes, the performance speedup that *Demarco* offers is more than three times.

Figures 4(a) to 4(d) show that the performance of the *Demarco* structures are comparable to the expected $O(1)$ complexity multilist-based priority queues, i.e. CQ and DCQ. Furthermore, Figures 4(e) and 4(f) demonstrates clearly that the *Demarco* structures outperform the CQs which have erratic performance for skewed distributions such as the Camel and Change. The reasons for their inferior performance are:

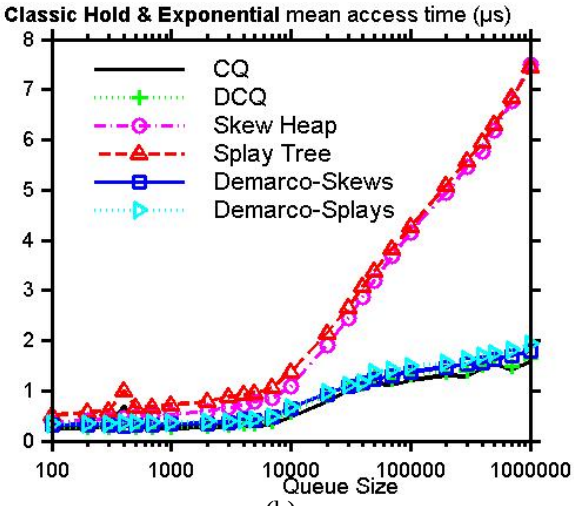
1. The CQ size-based resize triggers are an incompetent mechanism for handling skewed distributions. The triggers are too rigid to react according to the events distribution since a resize trigger occurs only if the queue size fluctuates by a factor of two [4]. It results in many events being enqueued into a few buckets with long sublists and many empty buckets. Long sublists make enqueue operations expensive since each enqueue entails a sequential search, whereas excessive traversal of empty buckets increases the process of dequeue operations. The DCQ incorporates two additional cost-based triggers, one for dequeue and another for enqueue operation. These additional triggers help to

reduce the instability faced by the CQ under Change distribution but however, the DCQ performs worse than the CQ for the Camel distribution.

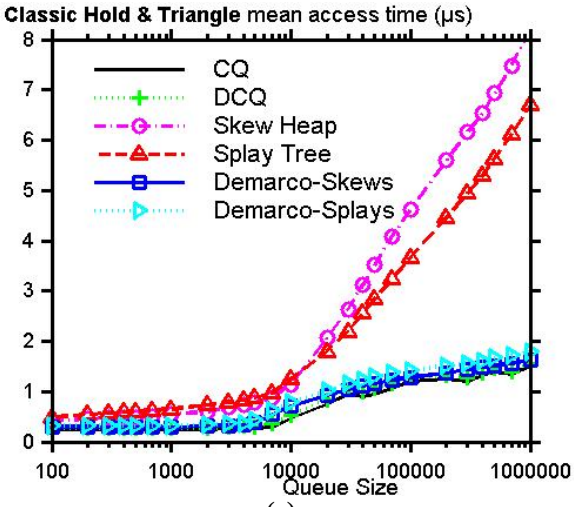
2. Sampling heuristic is inadequate to obtain good operating parameters, namely, the number of buckets and the bucketwidth, when skewed distributions are encountered. During each resize operation, the CQ samples at most the first twenty-five events. This is clearly too simplistic because for skewed distributions in which many events fall into some buckets, the inter-event time-gap of the first twenty-five events, which can span several buckets, and those in the few populated buckets may vary a lot. To simply increase the events sampled is not prudent unless it is uniform distribution. For most distributions, particularly skewed distributions such as the Camel, if we sample more events then take the mean or median, it is unlikely that it will be accurate since events are spread unevenly. Even if we sample the most populated bucket (i.e. in the DCQ), the DCQ also performs poorly because events in that bucket will have a small average time-gap whereas other events have widely diverse time-gaps. If the bucketwidth is updated to this small time-gap, there will likely be numerous empty buckets. Skipping these buckets will lead to inferior performance. Furthermore, sampling more events inevitably leads to higher overheads for each resize operation, affecting the CQ performance on a whole.



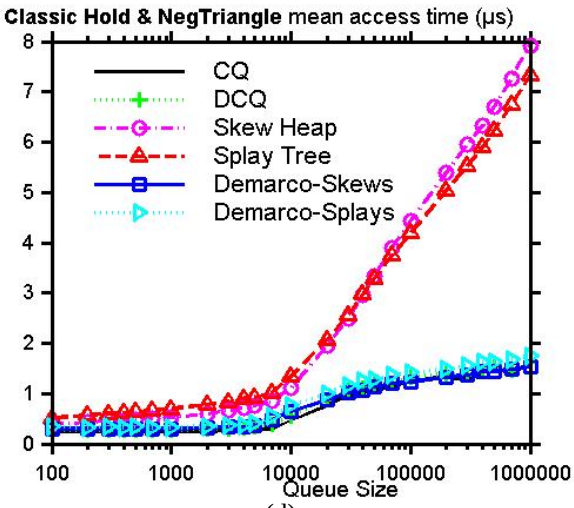
(a)



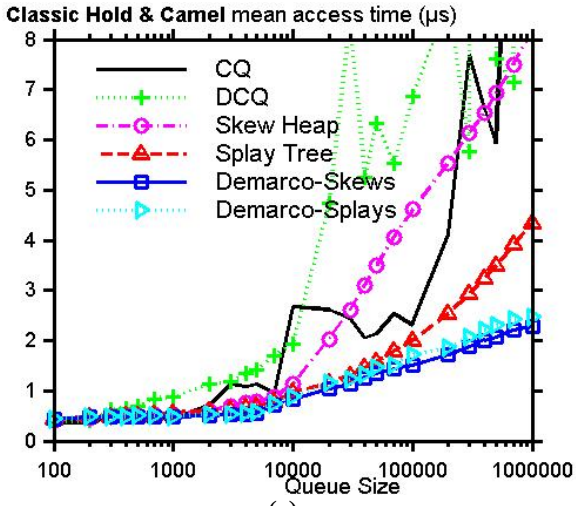
(b)



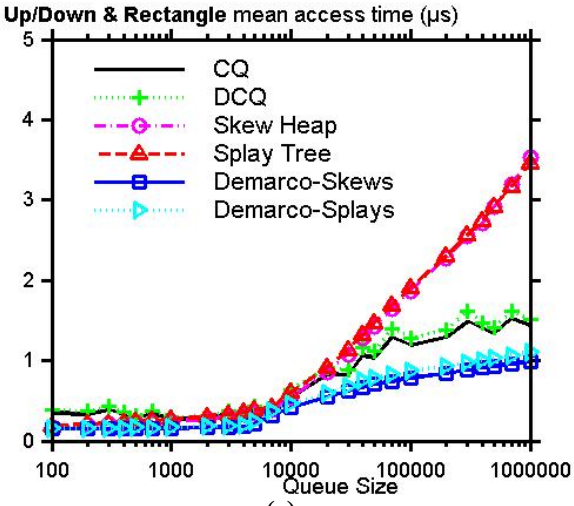
(c)



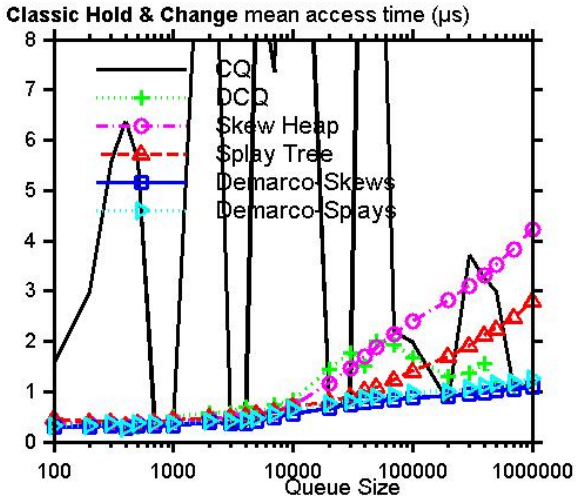
(d)



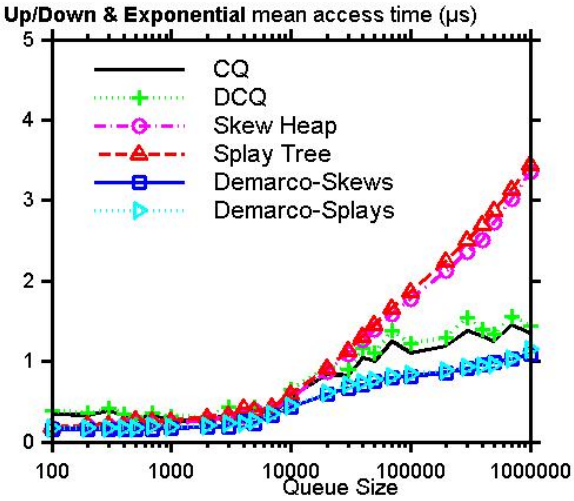
(e)



(a)

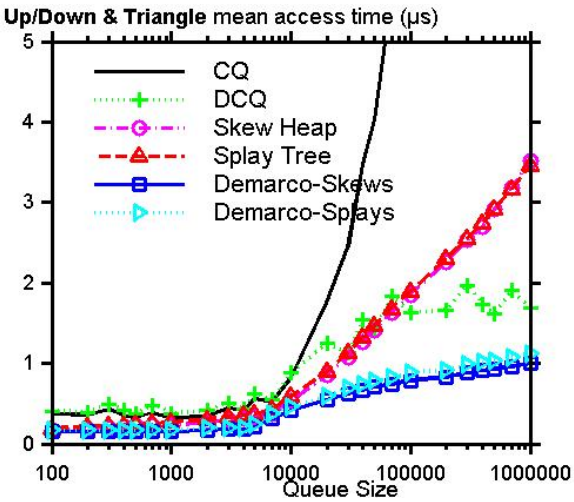


(f)



(b)

Figure 4: Performance graphs for Classic Hold model experiments.

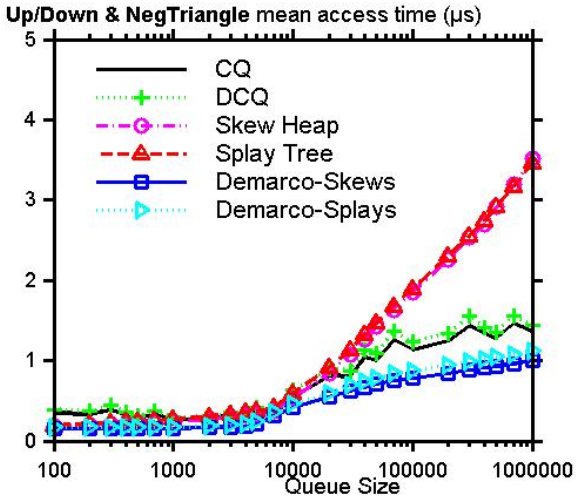


(c)

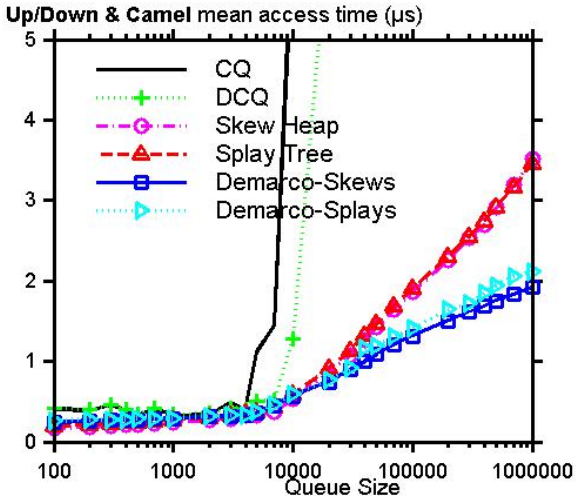
4.2 Transient-state experiments

The Up/Down model which tests the performance of priority queue structures during transient periods when the queue size fluctuates frequently, reveals the weaknesses of the CQs. Figures 5(a) to 5(d) show that the CQs experience several peaks and these suggest strongly that the resize operations found in the CQs can be costly since the CQs resize whenever the queue size fluctuates by factors of two. The form of triggers found in the CQs are clearly inflexible because even though the CQs can be performing well with its existing operating parameters, but because of their static triggers, they still have to resize whenever the queue size fluctuates by factors of two. Figures 5(e) and 5(f) again demonstrate that the CQs are sensitive to skewed distributions.

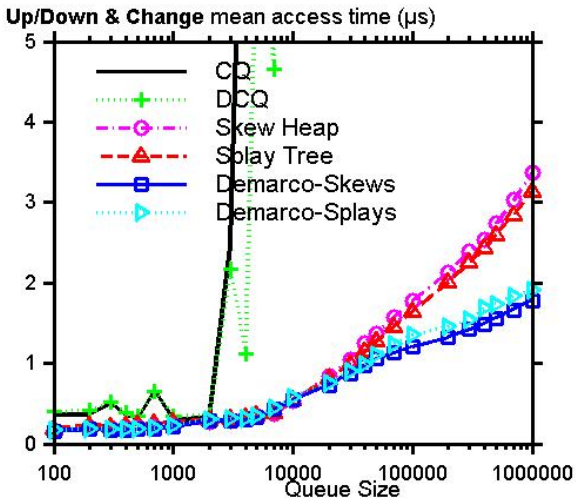
The *Demarco* structures outperform all the priority queues in all these experiments.



(d)



(e)



(f)

Figure 5: Performance graphs for Up/Down model experiments.

4.3 Overall performance comparison

This section illustrates numerically the performance speedup of the *Demarco* structures over the normal single tree-based priority queues. In addition, we compare the relative performance of the *Demarco* structures and tree-based priority queues versus the multilist-based CQ and DCQ.

Table 2: Speedup offered by the *Demarco* structure normalized over a single tree-based priority queue – comparison by priority increment distribution.

Distribution	*Demarco*-Skews	*Demarco*-Splays
Rectangle	2.99	2.61
Exponential	2.62	2.66
Triangle	3.01	2.46
NegTriangle	2.99	2.66
Camel	2.05	1.31
Change	1.91	1.39
Average	**2.60**	**2.18**

Table 3: Speedup offered by the *Demarco* structure normalized over a single tree-based priority queue – comparison by queue size.

Queue Size	*Demarco*-Skews	*Demarco*-Splays	Cost (MB)*
100	1.14832	1.27259	0.0016
200	1.21848	1.35236	0.0032
300	1.26669	1.39214	0.0048
400	1.32153	1.55857	0.0064
500	1.33875	1.47472	0.0080
700	1.37699	1.50817	0.0112
1000	1.42609	1.57609	0.0160
2000	1.45507	1.59857	0.0320
3000	1.58005	1.67455	0.0480
4000	1.60971	1.71499	0.0640
5000	1.59277	1.61879	0.0800
7000	1.33867	1.35547	0.1120
10000	1.38927	1.39643	0.1600
20000	1.78548	1.56020	0.3200
30000	1.99237	1.67113	0.4800
40000	2.17161	1.77727	0.6400
50000	2.30145	1.83629	0.8000
70000	2.50331	1.97654	1.1200
100000	2.69297	2.08904	1.6000
200000	3.00549	2.37097	3.2000
300000	3.15127	2.49080	4.8000
400000	3.21057	2.55460	6.4000
500000	3.33468	2.63459	8.0000
700000	3.48493	2.75407	11.2000
1000000	3.66322	2.88340	16.0000

*MB refers to one million bytes. Memory incurred is sixteen bytes per bucket and the number of buckets used is equal to the queue size tested.

Table 4: Relative performance for Exponential distribution
(normalized with respect to the fastest access time where 1.00 is the fastest).

Model	Queue Size	*Demarco*-Skews	*Demarco*-Splays	Skew Heap	Splay Tree	CQ	DCQ
Classic Hold	100	1.23	1.35	1.67	2.06	1.00	1.20
	1000	1.19	1.32	2.17	2.81	1.00	1.10
	10000	1.34	1.50	2.20	2.60	1.00	1.16
	100000	1.12	1.26	3.90	3.50	1.00	1.07
	1000000	1.05	1.17	5.33	4.77	1.00	1.03
	Average	1.19	1.32	3.05	3.15	1.00	1.11
Up/Down	100	1.00	1.06	1.15	1.22	2.31	2.57
	1000	1.00	1.04	1.56	1.73	1.76	1.94
	10000	1.00	1.12	1.31	1.45	1.33	1.54
	100000	1.00	1.12	2.39	2.43	1.52	1.64
	1000000	1.00	1.12	3.55	3.49	1.44	1.52
	Average	1.00	1.09	1.99	2.06	1.67	1.84
	Total Average	**1.09**	**1.21**	**2.52**	**2.61**	**1.34**	**1.48**

Table 5: Relative average performance for all distributions
(normalized with respect to the fastest access time where 1.00 is the fastest).

Model	Queue Size	*Demarco*-Skews	*Demarco*-Splays	Skew Heap	Splay Tree	CQ	DCQ
Classic Hold	100	1.07	1.14	1.31	1.61	1.59	1.00
	1000	1.15	1.26	1.76	2.15	1.00	1.41
	10000	1.00	1.10	1.55	1.70	5.32	1.23
	100000	1.00	1.12	3.27	2.59	1.20	1.81
	1000000	1.00	1.11	4.44	3.61	1.90	NA*
	Average	1.04	1.15	2.47	2.33	2.20	NA*
Up/Down	100	1.00	1.09	1.01	1.12	2.17	2.38
	1000	1.00	1.06	1.25	1.42	1.56	1.75
	10000	1.00	1.08	1.16	1.27	19.55	15.96
	100000	1.00	1.10	1.93	1.95	NA*	NA*
	1000000	1.00	1.10	2.67	2.63	NA*	NA*
	Average	1.00	1.09	1.60	1.68	NA*	NA*
	Total Average	**1.02**	**1.12**	**2.04**	**2.01**	NA*	NA*

* NA is meant that some of the access times are too high in at least one or more distributions. Thus the results are not considered in this comparison.

Table 2 shows that the speedup offered by the *Demarco* structure is more than two times, average over all queue sizes and distributions. Table 3 shows the speedup for each queue size, average under all the distributions. From the table, it is obvious that as the queue size increases, the speedup increases to more than three times for the Skew Heap and close to three times for the Splay Tree.

Tables 4 and 5 illustrate the relative performance of all the priority queues considered. The *Demarco*-Skews and *Demarco*-Splays outperform their tree-based counterparts and is generally more stable than the CQs.

4.4 Generality, sensitivity and cost-performance ratio of Demarco structures

Figures 6(a) and 6(b) shows the generality and insensitivity of *Demarco*-Skews under the various distributions and queue sizes (*Demarco*-Splays has similar graphs and is thus not included). Though the performance of *Demarco*-Skews may differ by as much as twice for different distributions, the complexity is still considered near $O(1)$. Furthermore, the graphs show that it is stable for all the distributions unlike the CQs which is near $O(n)$ for skewed distributions. This superior performance is made possible because of the four essential principles mentioned in Section 2.

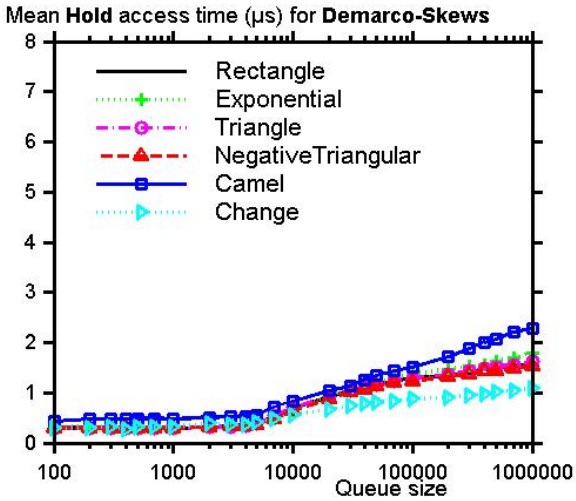


Figure 6: Performance graphs for *Demarco*-Skews.

We also consider the cost-performance ratio of employing Demaro on the tree-based priority queue. The additional memory allocation required to obtain a *Demarco* structure is as shown in Table 3. The bulk of memory requirement is the number of buckets allocated. Other memory variables such in *SecT* (e.g. *SecT_Min*, etc) take up an insignificant amount of memory and thus are not considered. In C programming language, a bucket is "`struct bucket { unsigned int count; struct event *evt; }`". Therefore, each bucket in *PriT* takes up eight bytes on a 32-bit hardware platform and sixteen bytes on a 64-bit platform. Table 3 assumes a higher cost of sixteen bytes per bucket and that the number of buckets used is equal to the queue size tested.

From Table 3, it is shown that if the queue size of a simulation is 100,000, the speedup expected for a *Demarco*-Skews is 2.69 and the cost of additional memory incurred is about 1.6MB, where MB refers to one million bytes. And for a one million queue size simulation, a *Demarco*-Skews performs 3.66 times better than a Skew Heap and the cost incurred is 16MB of shared memory. By today's standard, 16MB is considered affordable. This is even more so in the near future, when 64-bit workstations (which can accommodate more physical memory) begin to proliferate the desktop market.

5 Conclusion

Demarcate Construction is a new form of tree-based priority queues which employs the *demarcation* process. These new priority queues offer an average speedup of more than twice over the single tree-based counterparts and outperform the current expected *O*(1) Calendar Queues in many scenarios. Its generality in small to large queue sizes, insensitivity to priority increment distributions and low overhead costs, make it a superior priority queue for many applications such as the pending event set structure in discrete event simulations.

References

[1] Sleator, D. D. and Tarjan, R. E. (1985) Self-adjusting binary search trees. *Journal of the ACM 32*, 3 (July), pp. 652-686.

[2] Sleator, D. D. and Tarjan, R. E. (1986) Self-adjusting heaps. *SIAM Journal of Computing 15*, 1 (Feb.), pp. 52-69.

[3] Tarjan, R.E. (1985) Amortized computational complexity. *SIAM Journal on Algebraic and Discrete Meth. 6*, 2 (April), 306-318.

[4] Brown, R. (1988) Calendar queues: A fast *O*(1) priority queue implementation for the simulation event set problem. *Commun. ACM 24*, 12 (Dec.), 825-829.

[5] Oh, S., and Ahn, J. (1998) Dynamic calendar queue. In *Proceedings of the 32nd Annual Simulation Symposium*, pp. 20-25.

[6] Rönngren, R. and Ayani, R. (1997) A comparative study of parallel and sequential priority queue algorithms. *ACM Trans. Model. Comput. Simul. 7*, 2 (April), pp. 157-209.

[7] Schwetman, H. (1996) CSIM18 User's Guide. Austin, TX: Mesquite Software, Inc.

[8] Das, S., Fujimoto, R., Panesar, K., Allison, D., and Hybinette, M. (1994) GTW: a time warp system for shared memory multiprocessors. In *Proceedings of the 1994 Winter Simulation Conference*, pp. 1332-1339.

[9] Fall, K. and Varadhan, K. (2002) The ns Manual. UCB/LBNL/VINT Network simulator v2. http://www.isi.edu/nsnam/ns/.

[10] Stoica, I., Zhang, H., and Ng, T.S.E. (2000) A hierarchical fair service curve algorithm for link-sharing, real-time, and priority services. *IEEE/ACM Trans. Networking*, 8, 2 (Apr.), pp. 185-199.

[11] Hagai, A. and Patt-Shamir, B. (2001) Multiple priority, per flow, dual GCRA rate controller for ATM switches. IEEE Workshop on High Performance Switching and Routing 2001, pp. 169-174.

[12] Gomes F., S. Franks, B. Unger, Z. Xiao, J. Cleary, and A. Covington. (1995) SimKit: A high performance logical process simulation class library in C++. In *Proceedings of the 1995 Winter Simulation Conference*, pp. 706-713.

[13] Narlikar, G. and Zane, F. (2001) Performance modeling for fast IP lookups. In *Proceedings of the 2001 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pp. 1-12.

[14] Grinberg, D., Rajagopalan, S., Venkatesan, R., Wei, V. K. (1995) Splay trees for data compression, In *Proceedings of the sixth annual ACM-SIAM symposium on Discrete Algorithms*, pp. 522-530.

[15] Yugo, K. I. R., Moffat, A., and Ngai, C. H. A. (2002) Enhanced word-based block-sorting text compression. In *Proceedings of the twenty-fifth Australasian conference on Computer science*, pp. 129-137.

[16] Comfort, J. C. (1984) The simulation of a master-slave event set processor. *Simulation 42*, 3 (March), pp. 117-124.

[17] Dupuy, A., Schwartz, J., Yemini, Y. and Bacon, D. (1990) NEST: a network simulation and prototyping testbed, *Commun. ACM 33*, 10 (Oct.), pp. 63-74.

[18] Jones, D. W. (1986) An empirical comparison of priority-queue and event-set implementations. *Commun. ACM 29*, 4 (April), pp. 300-311.

[19] Rönngren, R., Riboe, J., and Ayani, R. (1993) Lazy queue: New approach to implementing the pending event set. *Int. J. Computer Simulation 3*, pp. 303-332.

[20] Park, S. K. and Miller, K. W. (1988) Random number generators: Good ones are hard to find. *Commun. ACM 31*, 10 (Oct.), pp. 1192-1201.

[21] Jones, D. W. (1989) Concurrent operations on priority queues. *Commun. ACM 32*, 1 (Jan.), pp. 132-137.

Fault-Free Maximal Submeshes in Faulty Torus-Connected Multicomputers

Seong-Moo Yoo
Electrical and Computer Engineering Department, University of Alabama in Huntsville
Huntsville, Alabama 35899, USA
e-mail: yoos@eng.uah.edu

Hee Yong Youn
School of Information and Communication Engineering, Sungkyunkwan University
Suwon, Korea
e-mail: youn@ece.skku.ac.kr

Hyunseung Choo
School of Information and Communication Engineering, Sungkyunkwan University
Suwon, Korea
e-mail: choo@ece.skku.ac.kr

Keywords: fault-free subsystem, maximal submeshes, reconfiguration, torus-connected multicomputers, virtual submeshes

Received: May 11, 2004

In a parallel computer system with faulty processors, it is highly desirable to reconfigure the system by eliminating the faulty ones and thereby restore the system to some operational state. In the reconfiguration maintaining the maximum size fault-free subsystem is the main problem. In this paper, we propose an efficient scheme for identifying maximum size fault-free submeshes in a faulty torus-connected multicomputer system. For this, the relations between two submeshes have been defined. Then we take two-phase approach. In the first phase, an efficient algorithm for determining maximal faulty submeshes in a faulty torus has been introduced. In the second phase, we have introduced a procedure to identify the maximal fault-free submeshes by removing all faulty submeshes from a whole torus. The time complexity of the proposed scheme is $O(N_f^3)$ where N_f is the number of faulty processors in a 2D torus. The proposed scheme can be utilized to the task allocation in 2D tori in the presence of failed nodes.

Povzetek: članek obravnava delovanje večprocesorskega sistema ob izpadu nekaj procesorjev.

1 Introduction

Fault-tolerance for the multiprocessor system is achieved either by using the workable portion of the system to emulate the whole machine with certain slowdown or by reconfiguring the machine to a smaller sized system after faults occur. Letting the fault-free part of a machine emulate entire system functionality tends to have limitations in practical use because slowdown could translate into a considerable performance loss. Hence, fault tolerance by reconfiguration is very important in such a large distributed computing environment for continued operation of the multiprocessor after the failure of one or more processors and/or links. Once the faulty elements have been identified, graceful degradation can be achieved by reconfiguring the multiprocessor and the distributed algorithm running on the multiprocessor. The algorithm is formulated to run on a single processor which would typically be the host or the resource manager in a commercial multiprocessor system.

In parallel computer systems consisting of a number of interconnected processors, incoming jobs are allocated to the subsystems of a required number of processors such as subcube for hypercube or submesh for mesh system, respectively. Due to its complexity, some processors in the entire structure can be defective, and then they need to be excluded from the allocation. In other words, all the processors in the allocated subsystem must be good. In the presence of failed nodes, thus, the system should be reconfigured to allow fast contiguous allocation of fault-free subsystems to incoming jobs.

Among several parallel computer systems, torus-connected multicomputer system has become popular, and it has become the base of many parallel computer systems such as the Tera Computer [1] and the Cray T3D [2]. Task allocation problem in 2D meshes has recently drawn a lot of attention [3-18]; however, the allocation problem in a torus system in the presence of faulty nodes has not. Reconfiguring a faulty system into smaller size systems has been investigated for the hypercube architecture [19-20], but little work has been reported for torus-connected system to the best of our knowledge. The reconfiguration problem in a torus-connected

multiprocessor system reduces to finding the maximum size fault-free submeshes.

In this paper, we propose a scheme which can effectively identify the maximum size fault-free submeshes in a two-dimensional (2D) torus. For this, the relations between any two submeshes in a 2D torus are first defined. Then two-phase approach is taken. In the first phase, the largest size faulty submeshes in a faulty torus are determined. In the second phase, largest size fault-free submeshes are identified by removing all faulty submeshes. The time complexity of the proposed scheme is $O(N_f^3)$, where N_f is the number of faulty processors in a 2D torus. The proposed scheme will be useful for task allocation in 2D torus with faulty processors, and it can be applied to a 3D torus system.

In the case of a real high performance parallel computer (numerical solver etc.), torus connection is crucial, high speed is needed, and faulty processors must be replaced. Usually, for an efficient parallel system, communication links have to operate with maximal speed. In the case of faulty processors, the original torus structure is lost, because some indirect communication channels have to be used, or new links introduced to bridge a faulty processor. In this paper, the whole algorithm is organized globally. It is supposed that the parallel system works in fact by the master-slave principle (computer farm). In such cases, the algorithm proposed could be useful even though the proposed algorithm has not considered faulty links and unconnected links that are left after removing faulty processors.

The rest of the paper is organized as follows. In Section 2, definitions and notation used throughout the paper are introduced. In Section 3, the procedures grouping the faulty processors into faulty submeshes are introduced. The procedure identifying the largest size fault-free submeshes is then presented. Finally, we conclude the paper in Section 4.

2 Definitions and Notation

In this section, we define meshes by introducing the index set. Refer to Figure 1. A node, $n = (x, y)$, $(0 \le x \le a\text{-}1 \wedge 0 \le y \le b\text{-}1)$, refers to a processor where a two-dimensional torus, $2DT(a, b) = \{(x, y) \mid (0 \le x \le a\text{-}1 \wedge 0 \le y \le b\text{-}1)\}$, is an $a \times b$ rectangular grid. Here a and b represent the width and height of the torus, respectively. A submesh, S (base, end) $= ((x_b, y_b), (x_e, y_e)) = \{(x, y) \mid (x_b \le x \le x_e \wedge y_b \le y \le y_e)\}$. w and h denote the width and height of S, respectively.

Definition 0: (*Neighbor*):

$n = (x_n, y_n)$, $m = (x_m, y_m)$.

$n \mid\text{-} m \Leftrightarrow (x_m = x_n + 1) \wedge (y_m = y_n)$: m is *right neighbor* of n, and n is *left neighbor* of m.

$n \perp m \Leftrightarrow (x_m = x_n) \wedge (y_m = y_n + 1)$: m is *upper neighbor* of n, and n is *lower neighbor* of m.

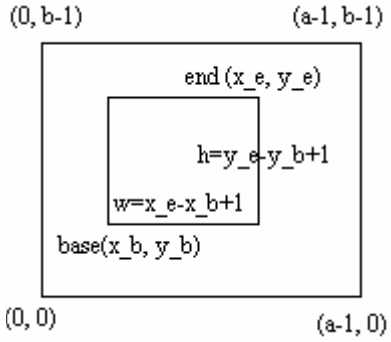


Figure 1. Index set for a 2D mesh and a submesh.

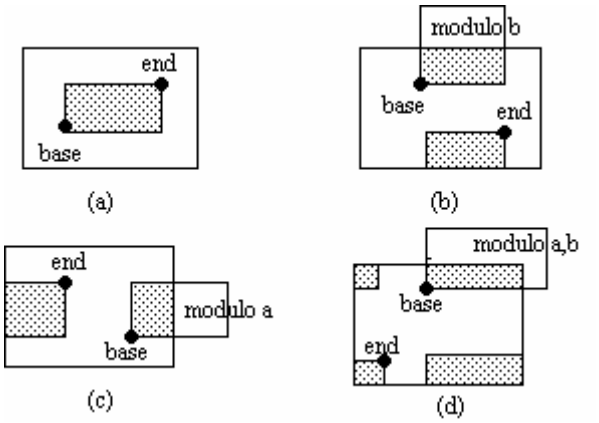


Figure 2. Four different virtual submeshes considering (base, end).

Definition 1: (*Virtual submesh*): Refer to Figure 2.

$S_v = ((x_b, y_b), (x_{ve}, y_{ve}))$ for

$S = ((x_b, y_b), (x_e, y_e))$, $x_e = x_{ve} \bmod a$,

$y_e = y_{ve} \bmod b$, $x_{ve} = \begin{cases} x_e & if \;\; x_b \le x_e \\ x_e + a & otherwise \end{cases}$,

$y_{ve} = \begin{cases} y_e & if \;\; y_b \le y_e \\ y_e + b & otherwise \end{cases}$.

Definition 2: *Separate* : $S_1 \Diamond S_2 \Leftrightarrow S_1 \bigcap S_2 = \phi$,

overlapped : $S_1 \mathrm{O}\, S_2 \Leftrightarrow S_1 \bigcap S_2 \neq \phi$.

Definition 3: *Covered* : $S_1 \cdot S_2 \Leftrightarrow S_1 \subseteq S_2$,

equivalent : $S_1 \equiv S_2 \Leftrightarrow (S_1 \subseteq S_2) \wedge (S_2 \subseteq S_1)$.

Definition 4: (*adjacent*)

$S_1 \parallel_r S_2 \Leftrightarrow \exists (n \in S_1, m \in S_2)$ [n |- m $\wedge$ $S_1 \Diamond S_2$]: S_2 is *right adjacent* of S_1, and S_1 is *left adjacent* of S_2.

$S_1 \parallel_u S_2 \Leftrightarrow \exists (n \in S_1, m \in S_2)$ [n $\perp$ m $\wedge$ $S_1 \Diamond S_2$]: S_2 is *upper adjacent* of S_1, and S_1 is *lower adjacent* of S_2.

Definition 5: (*Intersected submesh*) $S = S_1 \bigcap S_2$.

Definition 6: S is a *maximal submesh* in 2DT $\Leftrightarrow$ not $\exists\ S_2$ [$S_1 \cdot S_2$]. Maximal submeshes may either be separate or overlapping one another.

3 The Proposed Scheme

Our scheme finding maximal submeshes in a faulty 2DT consists of two phases. In the first phase, maximal faulty submeshes are determined by grouping the faulty processors. In the second phase, maximal fault-free submeshes are identified by removing all maximal faulty submeshes. For locating maximal faulty submeshes, a mechanism combining two submeshes need to be developed. We first consider that.

3.1 Maximal Faulty Submeshes

3.1.1 Combining Two Submeshes

Definition 7: *Combining* two submeshes, $S_1 + S_2$, is a procedure for identifying new maximal submeshes such that each of the new submesh consists of part or whole of S_1 and S_2. The details of the procedure are explained below.

There exist four cases in combining S_1 and S_2 according to the relative positions of them. Case 1: $S_1 \subseteq S_2 \vee S_2 \subseteq S_1$. Case 2: $S_1 \Diamond S_2 \wedge S_1 \nparallel S_2$. Case 3: $S_1 \parallel S_2$. Case 4: S_1 O $S_2 \wedge \neg(S_1 \subseteq S_2) \wedge \neg(S_2 \subseteq S_1)$. In Case 1, no new submesh is identified because $S_1 + S_2$ is S_1 or S_2 itself. In Case 2, no new submesh is identified either. In Case 3, one new submesh may be identified. S_3 if $S_1 \parallel_r S_2$ or S_4 if $S_1 \parallel_u S_2$ is generated as shown in Figure 3. The addresses of S_3 and S_4 can be referred to Table I. Note that the base and end points of a virtual submesh S_i are denoted by (x_{ib}, y_{ib}) and (x_{ie}, y_{ie}), respectively. In this case, S_1 should be deleted if $S_1 \subseteq S_3$ or S_4, and S_2 if $S_2 \subseteq S_3$ or S_4. In Case 4, no or two submeshes are identified. In Figure 4(a), two new submeshes, S_3 and S_4, are identified whose addresses are given in Table I. In Figure 4(b) and (c), no new submesh can be identified since $S_1 \equiv S_3$ and $S_2 \equiv S_4$. Therefore, in Case 4, S_3 and S_4 should be checked whether each of them is equivalent to either S_1 or S_2.

Table I. Addresses of new maximal submeshes obtained by combining two submeshes.

	S_3	S_4
x_b	min (x_{1b}, x_{2b})	max (x_{1b}, x_{2b})
y_b	max (y_{1b}, y_{2b})	min (y_{1b}, y_{2b})
x_e	max (x_{1e}, x_{2e})	min (x_{1e}, x_{2e})
y_e	min (y_{1e}, y_{2e})	max (y_{1e}, y_{2e})

When submeshes in 2DT are identified, we can do that for only separate ones or even overlapped ones. If only separate ones are identified, however, the largest submesh may not be able to be recognized. For example, refer to Figure 5.

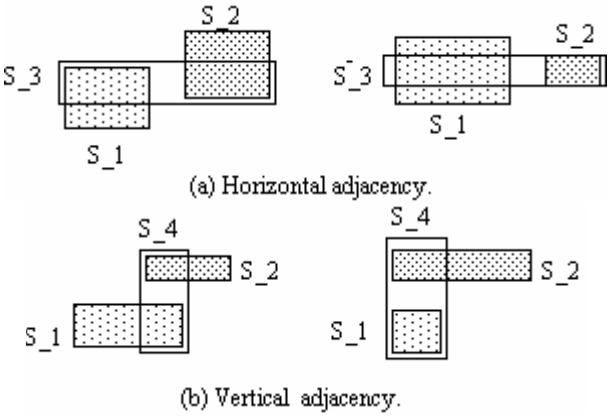


Figure 3. Generating new maximal submeshes when two submeshes are adjacent.

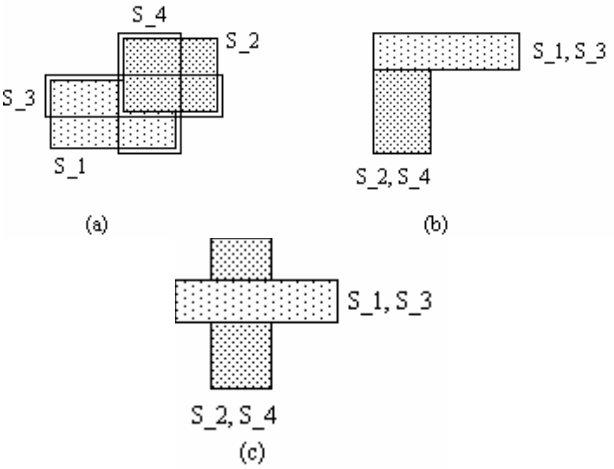


Figure 4. Generating new maximal submeshes when two submeshes overlap.

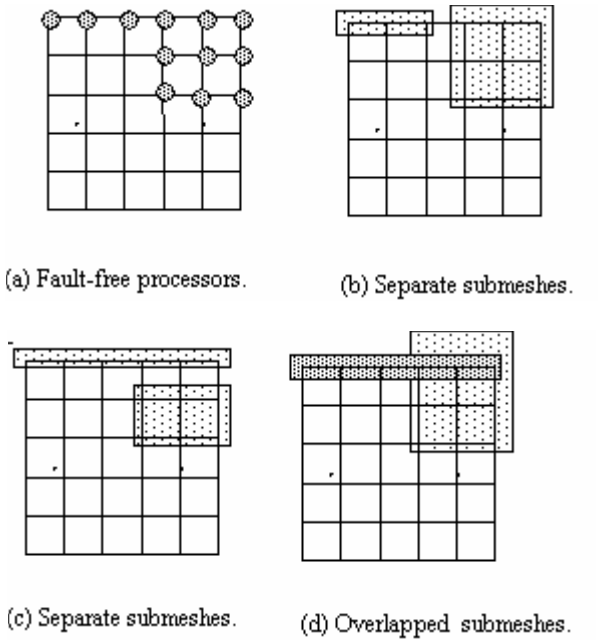


Figure 5. An example of recognizing the available largest size submeshes.

We want to identify maximal submeshes from 12 processors in Figure 5(a). The processors can be formed into two separate submeshes as shown in (b) and (c). Note that these are the two ways which can identify separate largest size submeshes. The way shown in Figure 5(b), however, cannot recognize the 6 × 1 submesh. Similarly, in Figure 5(c), the 3 × 3 submesh cannot be identified. On the contrary, if the identified submeshes are allowed to overlap as shown in Figure 5(d), all the available largest size submeshes can be recognized. Therefore, the proposed scheme identifies new submeshes which can overlap with other submeshes.

3.1.2 Finding Maximal Faulty Submeshes

Here we maintain a data structure *F*, an ordered list keeping faulty submeshes, for finding the maximal faulty submeshes. Initially any faulty processor is a submesh of one node itself, and *F* keeps the single node submeshes. These faulty submeshes are then combined into maximal faulty submeshes. They are combined first columnwise then rowwise. For this, all the faulty submeshes of a single node are inserted into *F* by the increasing order of *x*-coordinates of their bases, and then by the *y*-coordinates of the bases if the *x*-coordinates are same. If the base coordinates are the same, the end coordinates are considered. Next is the procedure for columnwise combining.

Procedure Column_combining ( )
$S_1 \leftarrow$ first submesh in F; $S_5 \leftarrow$Nil
Repeat
  $S_2 \leftarrow$ next submesh in F
  if ($x_{1b} = x_{2b}$)
    if ($y_{1b} = 0 \wedge S_1 \not\equiv S_5$) then $S_5 \leftarrow S_1$
    if ($S_1 \|_u S_2$)
      $S_3 \leftarrow S_1 + S_2$; delete S_1 and S_2 from F;
      insert S_3 into F
      if ($y_{3b} = 0$) then $S_5 \leftarrow S_3$
      $S_1 \leftarrow S_3$
    else
      $S_1 \leftarrow S_2$
  else
    if ($S_5 \neq$ Nil $\wedge S_1 \|_u S_5$)
      $S_3 \leftarrow S_1 + S_5$; delete S_1 and S_5 from F;
      insert S_3 into F
    $S_1 \leftarrow S_2$
    $S_5 \leftarrow$ Nil
Until all submeshes in F are considered

For example, in Figure 6(a), suppose that the 17 processors marked as *X* are faulty. Initially *F* contains 17 submeshes whose addresses are ((0, 0), (0, 0)), ((0, 2), (0, 2)), ((0, 3), (0, 3)), (0, 5), (0, 5)), ((1, 0), (1, 0)), ((1, 1), (1, 1)),..., and ((3, 5), (3, 5)). First consider two submeshes S_1((0, 0), (0, 0)) and S_2((0, 2), (0, 2)). Here S_5 is ((0, 0), (0, 0)). Since $S_2 \nparallel_u S_1$, S_1 and S_2 cannot be combined. Now consider S_1 ((0, 2), (0, 2)) and S_2((0, 3), (0, 3)). Since $S_1 \|_u S_2$, S_1 and S_2 are combined into S_3((0, 2), (0, 3)) and deleted from *F*. Next S_1((0, 2), (0, 3)) and S_2((0, 5), (0, 5)) cannot be combined. Consider S_1((0, 5), (0, 5)) and S_2((1, 0), (1, 0)). The *x* coordinates of bases of S_1 and S_2 are different. Therefore, we consider S_1((0, 5), (0, 5)) and S_5((0, 0), (0, 0)). Since $S_5 \|_u S_1$, those submeshes are combined into ((0, 5), (0, 0)) and deleted from *F*. The new submesh ((0, 5), (0, 0)) is inserted into *F*, and S_2((1, 0), (1, 0)) becomes new S_1. So far, ((0, 2), (0, 2)), ((0, 3), (0, 3)), ((0, 0), (0, 0)), and ((0, 5), (0, 5)) are deleted, and ((0, 2), (0, 3)) and ((0, 5), (0, 0)) are inserted. This procedure is continued and finally *F* contains ((0, 2), (0, 3)), ((0, 5), (0, 0)), ((1, 4), (1, 2)), ((2, 1), (2, 1)), ((2, 4), (2, 5)) and ((3, 1), (3, 5)) after the columnwise combining as shown in Figure 6(b). Note that the submeshes in *F* after columnwise combining have the following properties.

(1) The width of every submesh is 1.
(2) If S_1 is ahead of S_2 in F, either ($x_{1b} = x_{2b} \wedge y_{1e} < y_{2b}$) or ($x_{1e} < x_{2b}$) holds.
(3) All submeshes are separate.

Next the procedure for rowwise combining is given as follows.

Procedure Row_combining( )
$S_1 \leftarrow$ first submesh in F
$S_2 \leftarrow$ next submesh in F
Repeat
  if ($S_1 \|_r S_2 \ \vee \ S_2 \|_r S_1$)
    $S_3 \leftarrow S_1 + S_2$
    insert S_3 to F
    if $S_1 \subseteq S_3$, delete S_1 from F and $S_1 \leftarrow S_3$
    if $S_2 \subseteq S_3$, delete S_2 from F
  if there is next submesh of S_2 in F
    $S_2 \leftarrow$ the submesh
  else
    $S_1 \leftarrow$ next submesh of S_1 in F
    $S_2 \leftarrow$ next submesh of S_1 in F
Until no more next submesh of S_1 exists
  /* all submeshes in F are considered */

For example, suppose that the faulty processors are combined as shown in Figure 6(b) after the columnwise combining. Initially, all submeshes are arranged by the increasing order of the *y*-coordinates of their bases, and then by the *x*-coordinates of the bases if the *y*-coordinates are same. If the base coordinates are the same, the end coordinates are considered. Thus, the submeshes in Figure 6(b) are ordered as ((2, 1), (2, 1)), ((3, 1), (3, 5)), ((0, 2), (0, 3)), ((1, 4), (1, 2)), ((2, 4), (2, 5)), and ((0, 5), (0, 0)). First, considering S_1((2, 1), (2, 1)) and S_2((3, 1), (3, 5)) which are adjacent each other, new submesh ((2, 1), (3, 1)) is identified. Since ((2, 1), (2, 1)) $\subset$ ((2, 1), (3, 1)), ((2, 1), (2, 1)) is deleted from *F*, ((2, 1), (3, 1)) is inserted to *F*, ((2, 1), (3, 1)) is new S_1, and ((0, 2), (0, 3)) is new S_2. S_1 cannot be combined with S_2. Consider S_2((1, 4), (1, 2)). Here, a new submesh ((1, 1), (3, 1)) is identified and it is new S_1. The S_1 cannot be combined into any other remaining submesh ((2, 4), (2, 5)) or ((0,

5), (0, 0)) in F. Now new S_1 is ((3, 1), (3, 5)) and new S_2 is ((0, 2), (0, 3)). This procedure is continued, and final submeshes in F after rowwise combination become ((1, 1), (3, 1)), ((3, 1), (3, 5)), ((0, 2), (0, 3)), ((0, 2), (1, 2)), ((1, 4), (1, 2)), ((1, 4), (3, 5)), ((0, 5), (3, 5)), and ((0, 5), (1, 0)) as shown in Figure 6(c).

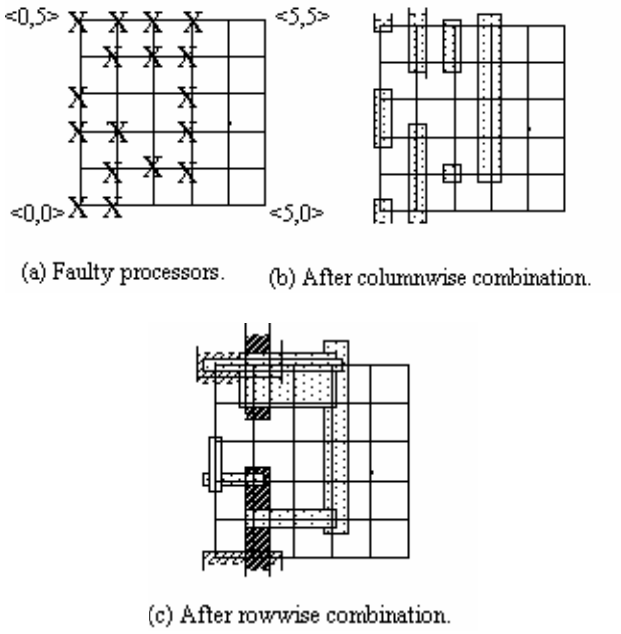


Figure 6. An example of finding faulty submeshes.

We analyze the time complexity of the procedure for finding maximal faulty submeshes as follows. Assume that N_f is the number of faulty processors in an $a \times b$ mesh system. The initial sorting of the faulty processors takes $O(N_f \log_2 N_f)$. Columnwise combining takes only $\theta(N_f)$ since each faulty processor is considered only once and deletion/insertion of a submesh tales $\theta(1)$ using a doubly-linked list. Rowwise combining takes $O(N_f^2)$ because all submeshes in F should consider all other submeshes in F in the worst case. Consequently, the time to find maximal faulty submeshes is $O(N_f^2)$.

We next show how the maximal fault-free submeshes are identified using the maximal faulty submeshes.

3.2 Maximal Fault-free Submeshes

Here we maintain another data structure *FF*, an ordered list keeping fault-free submeshes. The basic idea for identifying maximal fault-free submeshes is to chop off the portions of the fault-free submeshes which are part of maximal faulty submeshes. Initially, *FF* keeps one entire mesh. Then the faulty submeshes are removed from it one by one. In this section, the mechanism for removing faulty submeshes is first explained. Then the procedure for identifying maximal fault-free submeshes follows.

3.2.1 Removing Submeshes

Assume S_1 O S_2 and $S_9 \leftarrow S_1 \cap S_2$. Let k as the number of same coordinates in the addresses of S_1 and S_9. Recall that each submesh has four coordinates, x_b, y_b, x_e, and y_e. In Figure 7, the cases corresponding to five different values of k are shown. For example, in the case of Figure 7(c), $k = 2$ since $x_{1b} = x_{9b}$ and $y_{1b} = y_{9b}$, but $x_{1e} \neq x_{9e}$ and $y_{1e} \neq y_{9e}$.

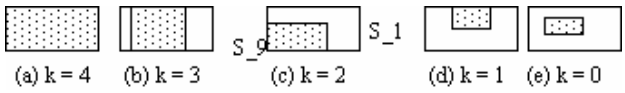


Figure 7. Five different values of k.

Definition 8: *Removing* S_2 from S_1, S_1 - S_2, is a procedure for identifying new maximal submeshes such that the new submeshes are the remainder of S_1 after excluding the processors belonging to S_2. S_1 - S_2 is the same as S_1 - S_9 where $S_9 \leftarrow S_1 \cap S_2$. If $S_1 \Diamond S_2$, S_1 - S_2 = S_1 and no change in S_1. If S_1 O S_2, new submeshes may be identified.

The details of the procedure for each case of k are explained below.

***k* = 4**: Since $S_1 \equiv S_9$, no submesh remains after removing S_9 from S_1.

***k* = 3**: There exist four cases as shown in Figure 8, Case 1: $y_{1e} \neq y_{9e}$, Case 2: $x_{1e} \neq x_{9e}$, Case 3: $y_{1b} \neq y_{9b}$, and Case 4: $x_{1b} \neq x_{9b}$. A new submesh S_3 is generated after removing S_9 from S_1. The address of S_3 is referred to Table II.

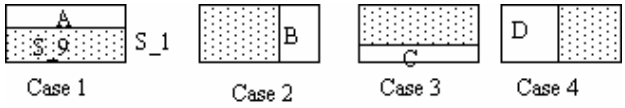


Figure 8. Four cases of location of intersected submesh when k = 3.

Table II. Address of new submesh from S_1 - S_9 when k = 3.

Case	S_3	x_{3b}	y_{3b}	x_{3e}	y_{3e}
1	A	x_{1b}	$y_{9e} + 1$	x_{1e}	y_{1e}
2	B	$x_{9e} + 1$	y_{1b}	x_{1e}	y_{1e}
3	C	x_{1b}	y_{1b}	x_{1e}	$y_{9b} - 1$
4	D	x_{1b}	y_{1b}	$x_{9b} - 1$	y_{1e}

***k* = 2**: There exist six cases as shown in Figure 9, Case 1: $x_{1b} = x_{9b} \wedge y_{1b} = y_{9b}$, Case 2: $x_{1b} = x_{9b} \wedge x_{1e} = x_{9e}$, Case 3: $x_{1b} = x_{9b} \wedge y_{1e} = y_{9e}$, Case 4: $y_{1b} = y_{9b} \wedge x_{1e} = x_{9e}$, Case 5: $y_{1b} = y_{9b} \wedge y_{1e} = y_{9e}$, and Case 6: $x_{1e} = x_{9e} \wedge y_{1e} = y_{9e}$. In each case, two new submeshes are generated after removing S_9 from S_1. The addresses of the new submeshes in Figure 9 are as follows; A, C, G: same as Case 1 in Table II; B, F, J: same as Case 2; D, E, K: same as Case 3; H, I, L: same as Case 4.

***k* = 1**: There exist four cases as shown in Figure 10, Case 1: $x_{1b} = x_{9b}$, Case 2: $x_{1e} = x_{9e}$, Case 3: $y_{1e} = y_{9e}$, and Case 4: $y_{1b} = y_{9b}$. In each case, three new submeshes are generated. The addresses of the new submeshes in Figure

10 are as follows; A, D, K: same as Case 1 in Table II; B, I, L: same as Case 2; C, F H: same as Case 3; E, G, J: same as Case 4.

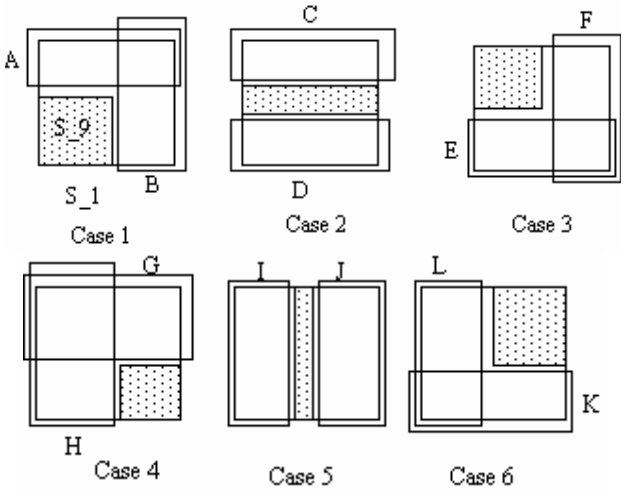


Figure 9. Six cases of location of intersected submesh when k = 2.

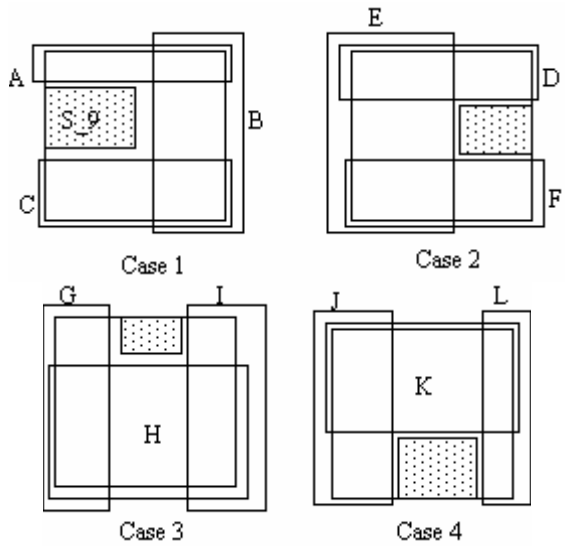


Figure 10. Four cases of location of intersected submesh when k = 1.

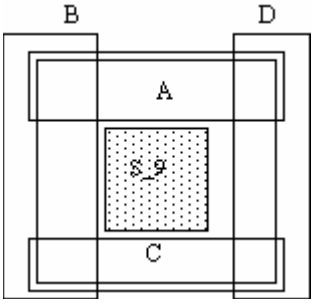


Figure 11. Location of intersected submesh when k = 0.

***k* = 0**: Four new submeshes are generated as shown in Figure 11. The addresses of the new submeshes in Figure 11 are as follows; A: same as Case 1 in Table II; D: same as Case 2; C: same as Case 3; B: same as Case 4.

3.2.2 Identifying Maximal Fault-free Submeshes

After the removing operation, the maximal fault-free submeshes are identified using the following procedure.

Procedure Collection_faultFree ( )
FF = { ((0, 0), (*a*-1, *b*-1)) }
F = set of faulty submeshes
while *F* is not empty
    $S_2 \leftarrow$ extract first submesh in *F*
    repeat
        $S_1 \leftarrow$ first submesh in *FF*
        if S_1 O S_2
            generate new submeshes from S_1 - S_2
                as explained in Section 3.2.1
            if any of those submeshes $\neg \subseteq$ any
                submesh in *FF*
                insert it into *FF*
    until all submeshes in *FF* are considered

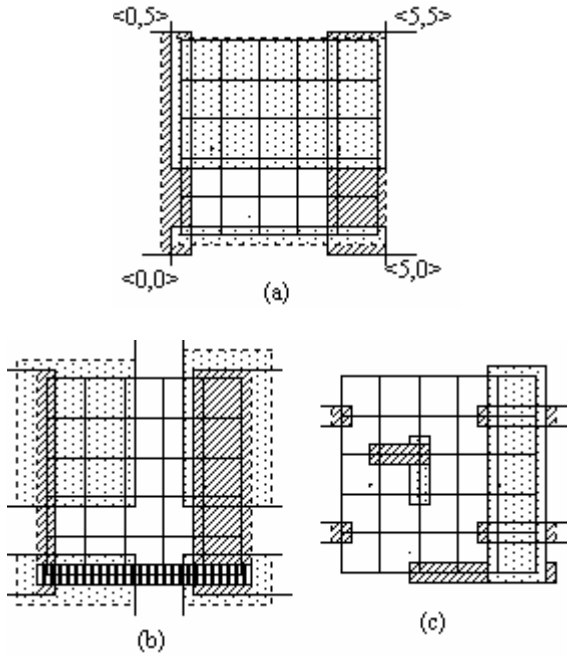


Figure 12. An example of removing faulty submeshes.

Recall that there exist 17 faulty processors (marked as 'X') in Figure 6(a) and those 17 processors are combined into 8 overlapped, faulty submeshes shown in Figure 6(c). Let us use the same example. There exist 19 fault-free processors (un-marked as 'X') in Figure 6(a). To identify the maximal fault-free submeshes from the 19 fault-free processors, the faulty submeshes in Figure 6(c) are removed from the whole torus, instead of directly combining the 19 fault-free processors. Refer to Figure 12. Initially, the whole torus ((0, 0), (5, 5)) exists. After removing ((1, 1), (3, 1)), the first faulty submesh in Figure 6(c), from the whole torus, two overlapped submeshes, ((0, 2), (5, 0)) and ((4, 0), (0, 5)), exist as shown in Figure 12(a). Next consider ((3, 1), (3, 5)), the second faulty submesh. It does not overlap with ((4, 0), (0, 5)) but overlaps with ((0, 2), (5, 0)). ((3, 1), (3, 5)) ∩

((0, 2), (5, 0)) = ((3, 2), (3, 5)), and ((0, 2), (5, 0)) – ((3, 2), (3, 5)) = ((0, 0), (5, 0)) and ((4, 2), (2, 0)). The remaining submeshes after the removing ((3, 1), (3, 5)) are ((4, 0), (0, 5)), ((0, 0), (5, 0)), and ((4, 2), (2, 0)) as shown in Figure 12(b). Similarly, remaining faulty submeshes, ((0, 2), (0, 3)), ((0, 2), (1, 2)), ((1, 4), (1, 2)), ((1, 4), (3, 5)), ((0, 5), (3, 5)), and ((0, 5), (1, 0)), are removed. Finally, as shown in Figure 12(c), there exist six fault-free submeshes, ((1, 3), (2, 3)), ((2, 0), (5, 0)), ((2, 2), (2, 3)), ((4, 0), (5, 5)), ((4, 1), (0, 1)), and ((4, 4), (0, 4)), which are maximal faulty-free submeshes.

We analyze the time complexity of the procedure to identify maximal fault-free submeshes. Initially, *FF* contains only one submesh and *F* contains $O(N_f)$ submeshes. The while loop executes $O(N_f)$ times and the repeat loop executes $O(N_f)$ times. Generation of new submeshes takes O(1) times. Comparison and insterstion take $O(N_f)$ times because *FF* is an ordered list. Therefore, the time to find maximal fault-free submeshes takes $O(N_f^3)$. Thus, the total time of the proposed scheme is $O(N_f^3)$. Recall that N_f is the number of faulty processors in an $a \times b$ mesh system.

4 Conclusion

In this paper, we have proposed a scheme which can efficiently identify the largest size fault-free submeshes in a 2D torus with faulty processors. The proposed scheme employs two phase approach for systematically find the desired fault-free submeshes. The main idea is that the largest size faulty submeshes are first identified, and then the portion of fault-free submeshes overlapped with the maximal faulty submeshes is excluded to find the largest size fault-free submeshes. For the effective manipulation of this process, the relative locations of any pair of submeshes in a 2D torus have also been defined. The time complexity of the proposed scheme is $O(N_f^3)$, where N_f is the number of faulty processors in a 2D torus.

Even though task allocation problem in 2D meshes has recently drawn a lot of attention, the allocation problem in a torus system in the presence of faulty nodes has not. We are currently developing a task allocation scheme in a faulty 2D torus based on the proposed reconfiguration scheme.

References

[1] Alverson et al., "The Tera computer system," Proc. 1990 Int'l Conf. on Supercomputing, pp. 1-6, 1990.

[2] R. E. Kessler and J. L. Schwarzmeier, "CRAY T3D: A new dimension for Cray research," in Proc. COMPCON, pp. 176-182, Feb. 1993.

[3] J. Ding and L.N. Bhuyan, “An adaptive submesh allocation strategy for two-dimensional mesh connected systems,” Int'l Conf. on Parallel Processing, pp. II-193-200, Aug. 1993.

[4] D.D. Sharma and D.K. Pradhan, “A fast and efficient strategy for submesh allocation in mesh-connected parallel computers,” Symp. on Parallel and Distributed Processing, pp. 682-689, Dec. 1993.

[5] S. M. Yoo, H.Y. Youn, and B. Shirazi, "An efficient task allocation scheme for 2D mesh architectures," IEEE Trans. on Parallel and Distributed Systems, pp. 934-942, September 1997.

[6] T. Liu, W.K. Huang, F. Lombardi, and L.N. Bhuyan, “A submesh allocation scheme for mesh-connected multicomputer systems,” Int'l Conf. on Parallel Processing, pp. II-159-163, August 1995.

[7] S. Bhattacharya and W.T. Tsai, “Lookahead processor allocation in mesh-connected massively parallel multicomputers,” Int'l Parallel Processing Symposium, pp. 868-875, April 1994.

[8] J. Upadhayay and P. Mohapatra, “An efficient processor allocation scheme for mesh connected parallel computers,” Symposium on Parallel and Distributed Processing, pp. 196-203, October 1995.

[9] W. Liu, V. Lo, and K. Windisch, “Non-contiguous processor allocation algorithms for distributed memory multicomputers,” Supercomputing, pp. 227-236, November 1994.

[10] G. Kim and H. Yoon, "On submesh allocation for mesh multicomputers: a best-fit allocation and a virtual submesh allocation for faulty meshes," IEEE Trans. on Parallel and Distributed Systems, pp. 175-185, February 1998.

[11] G.M. Chiu and S.K. Chen, "An efficient submesh allocation scheme for two-dimensional meshes with little overhead," IEEE Trans. on Parallel and Distributed Systems, pp. 471-486, May 1999.

[12] B.S. Yoo and C.R. Das, “A fast and efficient processor allocation scheme for mesh-connected multicomputers,” IEEE Trans. on Computers, 51 (1), pp. 46-60, Jan. 2002.

[13] L.D. de Cerio, M. Valero-Garcia, and A. Gonzalez, “Hypercube algorithms on mesh connected multicomputers,” IEEE Trans. on Parallel and Distributed Systems, 13 (12), pp. 1247–1260, Dec. 2002.

[14] D. Wang and J. Cao, “On optimal hierarchical configuration of distributed systems on mesh and hypercube,” Parallel and Distributed Processing Symposium, pp. 8, April 2003.

[15] K.H. Seo and S.C. Kim, “A dynamic processor management scheme on the reconfigurable meshes,” Int’l Conf. on Parallel and Distributed Computing, Applications and Technologies (PDCAT'2003), pp. 497-501, Aug. 2003.

[16] N.C. Wang and T.S. Chen, “Task migration in all-port wormhole-routed 2D mesh multicomputers,” Int’l Symp. on Parallel Architectures, Algorithms and Networks, pp. 123-128, May 2004.

[17] H.J. Ho and W.M. Lin, “A performance-optimizing scheduling technique for mesh-connected multicomputers based on real-time job size distribution,” Int’l Conf. on Parallel and Distributed Systems (ICPADS 2004), pp. 639-646, July 2004.

[18] A.L. Rosenberg, “On scheduling mesh-structured computations for Internet-based computing,” IEEE Trans, on Computers, 53(9), pp. 1176-1186, Sept. 2004.

[19] S. Latifi, “Distributed subcube identification algorithms for reliable hypercubes,” Information Processing Letters 38, pp. 315-321, June 1991.

[20] N.F. Tzeng and G. Lin, “Identifying maximal incomplete subcubes in faulty hypercubes,” Symp. on Parallel and Dist. Comp. Systems, pp. 186-193, Oct. 1994.

Distributing State Space for Parallel Computation of CTL Model Checking

Mustapha Bourahla
Computer Science Department, University of Biskra, Algeria
mbourahla@hotmail.com

Mohamed Benmohamed
Computer Science Department, University of Constantine, Algeria
ibnm@yahoo.fr

Keywords: Model Checking, Distributed Systems, Parallel Computation

Received: March 3, 2003

In this work we discuss the problem of performing distributed CTL model checking by splitting the given state space into several partial state spaces, and therefore the distribution of the transition relation. This work thus significantly extends the scope of properties that can be verified for very large designs. Each computer involved in the distributed computation owns a partial state space and performs a model checking algorithm on it. To be able to proceed, a CTL property is checked by the different processes and the results are combined to produce the final decision about the satisfaction of this CTL property. In this paper, we present the splitting algorithm of the state space. We will present also a new algorithm of the distributed CTL model checking whose correctness is formally proved.

Povzetek: članek analizira delovanje porazdeljenega CTL modela.

1 Introduction

The main aim in exploiting a distributed environment for model checking [5] is to extend the applicability of model checking algorithms to larger and more complex systems. Many sequential approaches have been proposed to deal with large state spaces, e.g. partial-order methods, symbolic verification, abstractions, and partial state space reasoning. Often these approaches do not suffice-time or space resources can still significantly limit the practical applicability [3]. Recently, a new promising method for increasing the memory capacity was introduced. The method uses the collective pool of memory modules in a network of processes. A parallel super computer, grid or a network of computers can provide extra resources needed to fight more realistic verification problems. Here we consider a network of workstations that communicate via message passing.

The important feature of algorithms running in a distributed environment is to solve the given task by distributing the data among the participating workstations with as small amount of coordination as possible. One of the main issues in distributing model checking algorithms is how to partition the state space (data) among the individual computers called here network nodes. There are two aspects that significantly influence the overall effectiveness of model-checking in the distributed environment: locality and (spatial) balance of the state space partition. Locality means that most of the state's descendants are assigned to the same node as the parent state, thus reducing communication and cooperation overhead. Balance means that each network node is assigned approximately the same number of states, thus achieving good speed-up.

The main idea of many distributed algorithms is similar: the state graph is partitioned among the network nodes, i.e., each network node owns a subset of the state space. The differences are in the way the state space is partitioned. Probabilistic techniques to partition the state space have been used e.g. in [2, 10, 14], and a technique which exploits some structural properties derived from the verified formula has been proposed in [1]. The model checking algorithm running on each network node has thus access only to a part of the entire system. Depending on the type of the algorithm, it communicates with other nodes to achieve the required (global) result.

The authors of [9, 16] have developed an approach to model checking of software which uses modularity. Their notion of a module differs from that used in modular model checking as understood for example in [7, 8, 13]. A module here is not a part of a whole system that runs in parallel with other modules (i.e. that contributes to the whole system in a multiplicative way), but a subset of a state space that originates from splitting the whole system in an additive way. It is defined by following the syntactical structure of the program. This notion of module has also been used in [11], where the system is split according to the semantics of the program. Besides this partition, the authors in [9, 16] have also defined the notion of an assumption function that represents partial knowledge about truth of formulas provided by the rest of the system (by other parts).

In this contribution we propose a new method of split-

ting the state space that is based on manipulations of Binary Decision Diagrams (BDDs) implementing this state space. Any transition system modeled as a Kripke structure, is implemented by a BDD. Our method will partition the original Kripke structure (the original BDD) by creating different BDDs that are linked by a way preserving the original behavior. Hence, these BDDs represent the different fragments of the original Kripke structure. The CTL property will be checked over the fragments of the Kripke structure in a parallel or a sequential way, where each fragment is used by a different process running the model checking algorithm on a workstation node. These processes will exchange information if necessary to make a decision about the satisfaction check. The exchange of information is based essentially on the links between the different fragments. Furthermore, we have modified the model checking algorithm in such a way that it can be run in a distributed environment.

The main idea is, once the system is partitioned, the Kripke structure on each network node can contain partial states and transitions between these states. These transitions can be unconditional transitions (their executions do not need exchange of information with other processes) or conditional where their executions will happen only after reception of information from other processes running on other nodes. This information represent the condition of these conditional transitions. In all cases we have also to take into account the associated communication complexity.

The rest of this paper is organized as follows: in Section 2, we present the definition of transition systems and their modeling as Kripke structures. Section 3 is devoted to our method of partitioning the state space. In Section 4, we present the language CTL and its semantics over a partitioned Kripke structure. The distributed CTL model checking algorithm and the proof of its correctness are presented in Section 5. At the end, in Section 6 we present a conclusion and related works.

2 Modeling Transition Systems

Our aim is to perform a model checking algorithm on a transition system which can model any concurrent system that can be described with a high level language, on a cluster of n workstations, called (network) nodes. In addition to the sequential case a partition algorithm is used to partition the state space among the nodes. After partitioning the state space, each node owns a part of the original state space. We will show that we can run distributed model checking on these parts of state spaces to verify CTL properties.

Definition 1 *A transition system M is a triple $M = (V, R, I)$ where $V = \{v_1, \ldots, v_n\}$ is a finite set of Boolean variables containing a sub-set of the atomic propositions (AP), $R \subseteq 2^V \times 2^V$ is a transition relation and $I \subseteq 2^V$ is a set of initial states.*

Definition 2 *A Kripke structure is a quintuple $K = (Q, R, s_0, AP, L)$ where:*

1. *Q is a non-empty (possibly infinite) set of states,*
2. *$R \subseteq Q \times Q$ is a total transition relation (i.e., $(\forall s : s \in Q : (\exists s' : s' \in Q : \langle s, s' \rangle \in R))$),*
3. *$s_0 \in Q$ is the initial state,*
4. *AP is a set of atomic propositions, and*
5. *$L : Q \rightarrow 2^{AP}$ is a state-labeling function.*

A Kripke structure may be viewed as labeled directed graph. The states are the graph's vertices, the transition relation defines the edges, the initial state is marked with a small incoming arrow, and each state is labeled with a set of atomic propositions.

A transition system is modeled with a Kripke structure, where the states in the Kripke structures (Q) are defined to be mapped to Boolean expressions composed of the variables from V describing the transition system. These Boolean expressions are defined in the transitions set of the original system. Note that, because the transition relation is required to be total, our model described with a Kripke structure has the same set of transitions defined by the relation R in the original transition system with the addition of loop transitions from and to vertices that don't have outgoing edges.

The labeling function $L : Q \rightarrow 2^{AP}$ is intended to associate with each state in Q an interpretation of the atomic propositions in AP. Thus, through L we know for each state $s \in Q$ which atomic propositions are true, namely those in $L(s)$, and which ones are false, namely those not in $L(s)$. The Kripke structures are implemented using the Reduced Order Binary Decision Diagrams (ROBDD or shortly BDD). These structures will be used as a base for our algorithms.

Example 1 *The Kripke structure of the transition system defined by the triple $M = (V = \{a, b, c\}$, $R = \{\langle a \wedge b \wedge c, a \wedge \bar{b} \wedge \bar{c} \rangle, \langle a \wedge b \wedge c, \bar{a} \wedge b \wedge \bar{c} \rangle, \langle a \wedge \bar{b} \wedge \bar{c}, \bar{a} \wedge b \wedge \bar{c} \rangle, \langle a \wedge \bar{b} \wedge \bar{c}, \bar{a} \wedge \bar{b} \wedge c \rangle, \langle \bar{a} \wedge b \wedge \bar{c}, a \wedge b \wedge \bar{c} \rangle, \langle \bar{a} \wedge \bar{b} \wedge c, a \wedge b \wedge \bar{c} \rangle\}$, $I = a \wedge b \wedge c)$, is shown in Figure 1, where s_0 is the initial state of this Kripke structure. We assume $AP = \{b\}$.*

3 Partitioning the Kripke Structure

The Kripke structure (which is represented by a BDD) modeling a transition system will be partitioned using the algorithm defined below (Algorithm 1). To partition any Kripke structure, we have to define a list of clusters C of Boolean variables from the set of variables V. Then, each cluster of variables from C, is used to generate a corresponding Kripke sub-structure. We note that the variables in V are ordered. The clusters C should be also ordered which means that $C_i \prec C_j$ if $\forall v_i : v_i \in C_i$ and $\forall v_j : v_j \in C_j$ then $v_i \prec v_j$. Each Kripke sub-structure

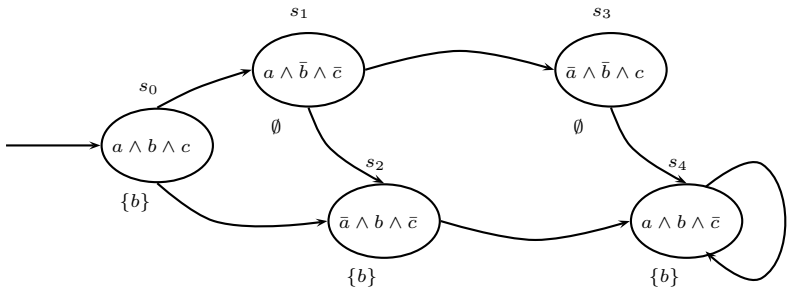


Figure 1: Kripke structure of the transition system M

K_i is composed of states that can be mapped to Boolean expressions B of the variables in C_i with the condition that $\exists B'$ such that $B \in B'$ and B' is the Boolean expression using the whole set V, of a state s_i in the original Kripke structure K. The composition of the generated Kripke sub-structures $K_i, i = 0, 1, \cdots, n$ should be equal to the original Kripke structure K.

Algorithm 1 `Partionning the Kripke structures`

BDD partition(BDD K, G, H; ClustersList C)
{
  if ($C \neq nil$) {
    if (K == $true$) return G
    if (K == $false$) return H
    x = min($root(K)$, $root(G)$, $root(H)$)
    if ($x \in CurrentCluster(C)$) {
      T = partition(K_x, G_x, H_x, C)
      E = partition($K_{\bar{x}}$, $G_{\bar{x}}$, $H_{\bar{x}}$, C)
      if T == E return T
      return createNode("x", T, E)
    } else {
      K_i = partition(K, G, H, $NextCluster(C)$)
      if ($\exists P \in FragmentsList \mid P == K_i$)
        return createNode(P.Label, nil, nil)
      else {
        InsertFragment(K_i)
        return createNode("K_i", nil, nil)
      }
    }
  } else return nil
}

The Kripke sub-structures resulting from the partition of the given Kripke structure, are called fragments.

Definition 3 *A Kripke structure $K_i = (Q_i, R_i, s_{0i}, AP_i, L_i)$ is a fragment of a Kripke structure $K = (Q, R, s_0, AP, L)$ if*

- $\forall s : s \in Q_i \Rightarrow \exists s' : s' \in Q \mid s \wedge s' = s'$
- $s_{0i} \wedge s_0 = s_0$
- $\forall \langle s_1, s_2 \rangle \in R_i \Rightarrow \exists \langle s'_1, s'_2 \rangle \in R \mid (s_1 \wedge s'_1 = s'_1) \wedge (s_2 \wedge s'_2 = s'_2)$
- $AP_i \subseteq AP$
- $L_i : Q_i \rightarrow 2^{AP_i} \mid \forall s : s \in Q_i : (\exists s' : s' \in Q : ((s \wedge s' = s') \wedge (L_i(s) \subseteq L(s'))))$.

This algorithm is using a list of fragments $FragmentsList$ to store the generated fragments of the original Kripke structure. By this algorithm, we can partition a complete Kripke structure K to many fragments K_i. Each K_i is a Kripke structure of a sub-system. Once the given system is partitioned, the resulting Kripke structures $K_0, \cdots, K_n$ can have links between them which make the sub-systems dependant. A fragment K_i of K is a sub-structure of K satisfying the property that every state in K_i can have unconditional and/or conditional successor states in K_i.

Then, a transition in the system represented by the fragment K_i can be unconditional or conditional transition. Formally, if $\exists K_i$ and $\exists \langle s_1, s_2 \rangle : \langle s_1, s_2 \rangle \in R_i$ (R_i is the transition relation of K_i), we denote the unconditional transition by $s_1 \rightarrow s_2$ which means a transition from the state s_1 to the state s_2 can occur without any reference to the other fragments (its execution is independent). By contrast, the conditional transition has the form $s_1 \xrightarrow{process_j} s_2$, where $process_j$ is a link label to another Kripke structure K_j.

This will be handled by communications between the processes running the different sub-structures. This means that a process will be suspended on a conditional transition until reception of information from the process running the Kripke sub-structure indicated by the label. There is a particular Kripke sub-structure which is not referenced by any transition in the other sub-structures. We call this the root Kripke sub-structure.

A path π in a Kripke structure K from a state s_0 is a sequence $\pi = s_0 s_1 \cdots$ such that $\forall i \geq 0 : s_i, s_{i+1} \in Q$ and $\langle s_i, s_{i+1} \rangle \in R$. An equivalent path can be computed using the fragments of K. Figure 2 shows the partition of the Kripke structure presented in Figure 1 after the execution of the following call: InsertFragment(partition(K, $true$, $false$, $\{\{a\}, \{b\}, \{c\}\}$)).

There are three clusters of variables $(\{\{a\}, \{b\}, \{c\}\}$. Each Kripke structure K_i represents a sub-system M_i. The sub-systems M_i can be executed in sequence or in parallel. These Kripke sub-structures can be represented as BDDs (It suffice just to remove the transition labels). So, for each

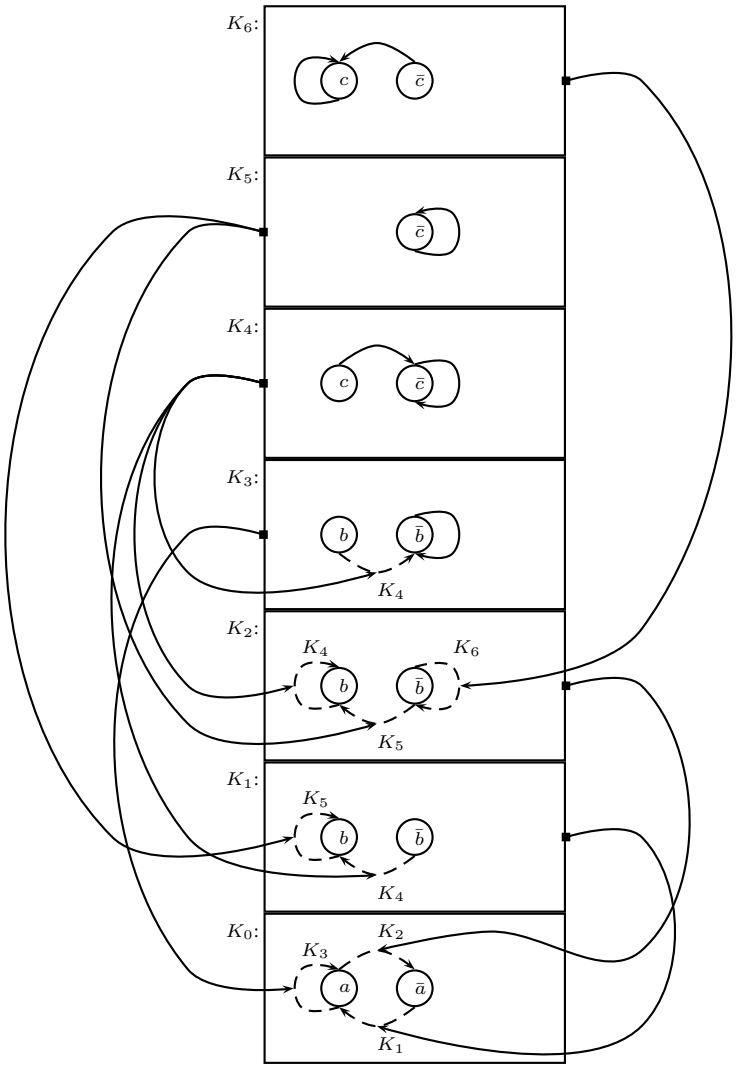


Figure 2: Fragments of K with their links

state we realize a formula of the form $F = f \wedge \bigvee_{i=1}^{m} g_i$, where f is the state formula and the sub-formulas g_i are the labels on its outgoing edges.

The transitions drawn by the dashed edges have a label indicating which sub-system to communicate with ($receivefrom()$) to get information (the condition) before this transition can take place (it is waiting for an information). We should remark that the number of generated Kripke sub-structures depends on the number of the specified clusters of variables and the original Kripke structure. Also, some Kripke sub-structures can have the same set of states but different transitions. This is very important to reduce the communication complexity and to improve the performance of the model checking. The advantage of representing any transition system with a fragmented and therefore distributed Kripke structure that we can execute this transition system in a sequential or in parallel manner.

The main goal of our distributed algorithm is to reduce the memory requirement. In symbolic model checking, pre-image is one of the operations with the highest memory requirement. Given a set of states S, pre-image computes $pred(S)$, which is the set of all predecessors of states in S. The pre-image operation can be described by the formula $pred(S) = \{s' \in Q \mid \exists s \in S : R(s', s)\}$. It is easy to see that the memory requirement of this operation grows with the sizes of the transition relation R and the set S. Furthermore, intermediate results sometimes exceed the memory capacity even when $pred(S)$ can be held in memory.

Our distributed algorithm reduces memory requirements by restricting each of the computed sets of states to a fragment. This takes care of the S parameter of pre-image, and the reduction of the transition as well. The following algorithm is the parallel version of the pre-image function $pred$.

Algorithm 2 *the pre-image function*

2^Q *pred(Kripke K,* $2^Q S$*)*
{
$res = \{s \wedge receivefrom(process_i) \mid$
$\exists t.s \xrightarrow{process_i} t \wedge t \in S\} \cup \{s \mid \exists t.s \rightarrow t \wedge t \in S\}$
for each process $process_i \neq pid$
sendto($process_i$*, res)*
return res
}

For each Kripke sub-structure K_i, we associate a process to calculate the pre-image function $pred$ in parallel with the other processes. The image function $succ$ can be defined by the same way. All the processes $process_i, i = 0, \cdots, n$ execute $pred$ once in parallel and then they begin communication to exchange information until there are no information to exchange which indicates the end of the function pre-image $pred$. The global result is the pre-image result of the process running the root Kripke sub-structure.

Example 2 *In this example (Figure 3), we compute the pre-image of the states characterized by the expression* $S = \bar{a} \wedge b \wedge \bar{c}$*. We observe that the computation is parallel where the function pre-image* $pred$ *is executed simultaneously on all the Kripke sub-structures by different processes* $P_0, \cdots, P_6$ *located on different nodes. If a process has a link label to another process, it will send a message to get the result of its execution. This communication will continue until all the information requested has been delivered. The result of the global function pre-image is the result of the process running the root Kripke sub-structure* K_0*.*

To reconstruct the original Kripke structure, it suffices to remove the conditional transitions by making them unconditional transitions. To do that we replace each conditional transition by transitions which are the product of this conditional transition and all the unconditional transitions in the fragment indicated by the link label on the conditional transition. The source states and the target states are also composed to get new source and target states. The resulting new transitions will inherit the attribute unconditional. This process will be iterated many times until there will be no conditional transitions.

Algorithm 3 *Reconstructing the original Kripke structure*

Kripke Reconstruct(Kripke K)
{
while $\exists \tau = s_1 \xrightarrow{K_j} s_2 \in K$ *do {*
if K_j *contains conditional transitions then*
K_j *= Reconstruct(*K_j*)*
for all transitions $s'_1 \rightarrow s'_2 \in K_j$ *do {*
$Q = Q \cup \{s_1 \wedge s'_1, s_2 \wedge s'_2\}$
$R = R \cup \{\langle s_1 \wedge s'_1, s_2 \wedge s'_2 \rangle\}$
}
}
}

To get the original Kripke structure from its fragments, we have just to execute the call $Reconstruct(K_0)$.

Lemma 1 *The produced Kripke structure by removing the conditional transitions from the fragments of a Kripke structure, is the same original Kripke structure used as input to the partition algorithm.*

We have illustrated the parallel execution which is our goal in addition to solving the explosion problem. The sequential execution is also possible using the same structure. In the next section, we will present the CTL semantics over a transition system represented by fragments of a Kripke structure.

4 CTL Model Checking

Computational Tree Logic (CTL) is the most popular temporal logic introduced for formal verification with model checking [9, 10]. This logic is formally based on models where at each moment there may be several different possible futures (branching temporal logic). Its semantics is a tree of states, each path in the tree is intended to represent a single possible computation of the system. The tree itself thus represents all possible computations. The CTL operators allow the expression of properties of some or all computations of a system.

Definition 4 *(Syntax of CTL). The language of CTL is defined by the following abstract syntax:*

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \exists X\varphi \mid \exists G\varphi \mid \varphi \exists U \varphi$$

where p ranges over the set of atomic propositions AP*,* X *is pronounced "next",* $\exists$ *"for some path",* $\forall$ *"for all paths" and* U *"until".*

There are two types of operators, state operators X, F and G, and path operators $\exists$ and $\forall$. All CTL temporal operators are composed of a path operator followed by a state operator. This grammar is not given in its most succinct form and there exist equivalence rules to express the same formula with different operators. In practice, by using this equivalence rules, a formula can be written such that the negation appears only at the level of atomic propositions. Such a form of a formula is known as Positive Normal Form (henceforth PNF form) [6].

A CTL-model $\mathcal{M}$ is represented as a Kripke structure ($\mathcal{M} = (Q, R, s_0, AP, L)$). A path ρ is an infinite sequence of states $s_0 s_1 s_2 \cdots$ such that $\langle s_i, s_{i+1} \rangle \in R, \forall i \geq 0$. $\rho[i]$ denotes the $(i+1)$-th element of ρ. The set of paths starting in state s of the model $\mathcal{M}$ is defined by $\Pi_{\mathcal{M}}(s) = \{\rho \mid \rho[0] = s\}$.

For any CTL-model $\mathcal{M} = (Q, R, s_0, AP, L)$ and state $s \in Q$, there is an infinite computation tree with root labeled s such that (s', s'') is an arc in the tree if and only if $(s', s'') \in R$. A state s for which $p \in L(s)$ is sometimes called a p-state. ρ is called a p-path if it consists solely of p-states.

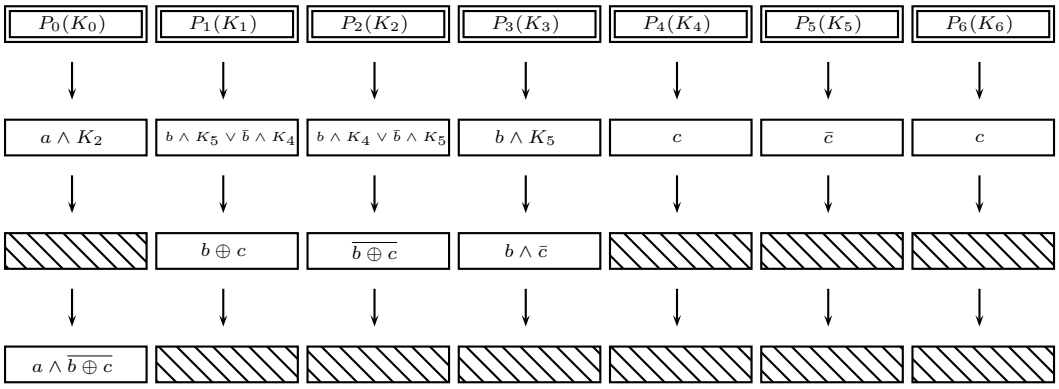


Figure 3: Parallel computation

Definition 5 *(Semantics of CTL). The semantics of CTL are defined by a satisfaction relation between a model $\mathcal{M}$, one of its states s, and a formula φ. $s \models_{\mathcal{M}} \varphi$ if and only if φ is valid in state s of model $\mathcal{M}$. The satisfaction relation is defined by:*

- $s \models_{\mathcal{M}} p$ *iff* $p \in L(s)$
- $s \models_{\mathcal{M}} \neg p$ *iff not* $(s \models_{\mathcal{M}} p)$
- $s \models_{\mathcal{M}} \varphi \wedge \psi$ *iff* $(s \models_{\mathcal{M}} \varphi) \wedge (s \models_{\mathcal{M}} \psi)$
- $s \models_{\mathcal{M}} \exists X\varphi$ *iff* $\exists \rho \in \Pi_{\mathcal{M}}(s).\rho[1] \models_{\mathcal{M}} \varphi$
- $s \models_{\mathcal{M}} \exists G\varphi$ *iff* $\exists \rho \in \Pi_{\mathcal{M}}(s).\forall j \geq 0.\rho[j] \models_{\mathcal{M}} \varphi$
- $s \models_{\mathcal{M}} \varphi \exists U \psi$ *iff* $\exists \rho \in \Pi_{\mathcal{M}}(s).(\exists j \geq 0.\rho[j] \models_{\mathcal{M}} \psi) \wedge (\forall k, 0 \leq k < j.\rho[k] \models_{\mathcal{M}} \varphi)$

Given a CTL-formula φ and a CTL-model $\mathcal{M}$ with a finite set of states (Q), the model checking algorithm computes the set of states for which φ is valid. This set is denoted $Sat(\varphi)$, and is computed in a recursive way, i.e. by computing for each sub-formula ψ of φ the set $Sat(\psi)$. In order to decide whether $s \models_{\mathcal{M}} \varphi$ we just have to check whether state s is a φ-state, i.e., whether $s \in Sat(\varphi)$.

Theorem 1 *The system M represented by the Kripke structure K and the system M' represented by its fragments $K_i, 0 \leq i \leq n$ are bi-similar.*

Proof. We have to prove that M simulates M' and M' simulates M. formally,

$$M \sim M' \Leftrightarrow (\forall \langle s_1, s_2 \rangle \in R \rightarrow \exists \langle \widetilde{s}_1, \widetilde{s}_2 \rangle \in \widetilde{R}) \wedge$$

$$(\forall \langle \widetilde{s}_1, \widetilde{s}_2 \rangle \in \widetilde{R} \rightarrow \exists \langle s_1, s_2 \rangle \in R),$$

where $\widetilde{s}_1$ and $\widetilde{s}_2$ are global states in M' that are computed from s_1 and s_2 respectively by the partition algorithm. $\widetilde{R}$ is the global transition relation of M'. We assume there are $n+1$ processes each one is running a fragment from the $n+1$ Kripke sub-structures. By construction s_1 has a corresponding global state in M', which is defined by $\widetilde{s}_1 = \bigwedge_{i=0}^{n} s_1^i$. Then it exists a next state of s_1 that also is represented with the same form by the next state of $\widetilde{s}_1$ which can be computed by the global transition relation $\widetilde{R}$.

$$\exists \widetilde{s}_2 : \widetilde{R}(\widetilde{s}_1, \widetilde{s}_2) =$$

$$\exists s_2^0 : R_0(s_1^0, s_2^0 \wedge receivefrom()) = \cdots =$$

$$R_0(s_1^0, s_2^0 \wedge \bigwedge_{j=1}^{n} receivefrom()) = R_0(s_1^0, \bigwedge_{j=0}^{n} s_2^j),$$

where $R_i, i = 0, \cdots, n$ are the transition relations of the corresponding fragments K_i. Each $process_j$ $j = 1, \cdots, n$ will send its image state s_2^j computed by using the local transition relation R_j, to the root process. The final image result is $\bigwedge_{j=0}^{n} s_2^j$. We remark that by construction s_2 is in each s_2^j then, $s_2 \in \bigwedge_{j=0}^{n} s_2^j$.

On the other hand, for $\forall \widetilde{s}_1$ we assume there exists s_1 which is in $\widetilde{s}_1$. By definition of Kripke structure it exists s_2 such that $R(s_1, s_2)$. By construction s_2 has a corresponding global state in M', which is defined by $\widetilde{s}_2 = \bigwedge_{i=0}^{n} s_2^i$. By construction of the partial transition relations during the partition process, it is easy to verify that $\widetilde{R}(\widetilde{s}_1, \widetilde{s}_2)$ is true. □

To define the semantics of CTL formulas over Kripke structures composed with fragment sub-structures we need to adapt the standard semantic definition. CTL is usually interpreted over complete structures, while our structures are typically non-complete. By theorem 1, we have the following equivalences in the definition of semantics.

Lemma 2 *(Semantics of the distributed system). Let φ be a CTL formula, and K_i ($0 \leq i \leq n$) are the fragments of K, computed using the partition algorithm. Then, we have this equality*

$$Sat(K, \varphi) = \bigcap_{i=0}^{n} (Sat(K_i, \varphi)).$$

5 Distributed CTL Model Checking

Each fragment K_i is managed by a separate process P_i. These processes are running in parallel on each node. Each

Algorithm 4 *Distributed CTL Model Checking*

2^Q *pSat(Kripke* K_i*, CTL* φ*)*
{
  switch(φ*) {*
    case p:
      $res = \{s \wedge receivefrom(P_j) \mid p \in L_i(s) \wedge \exists t \in Q_i.t \xrightarrow{P_j} s \in R_i\} \cup$
        $\{s \mid p \in L_i(s) \wedge \nexists t \in Q_i \wedge (\forall j : 0 \leq j \leq n.t \xrightarrow{P_j} s \in R_i)\}$
      sendtoOtherProcesses(res)
    case $\neg p$*:*
      $res = \{s \wedge receivefrom(P_j) \mid p \notin L_i(s) \wedge \exists t \in Q_i.t \xrightarrow{P_j} s \in R_i\} \cup$
        $\{s \mid p \notin L_i(s) \wedge \nexists t \in Q_i \wedge (\forall j : 0 \leq j \leq n.t \xrightarrow{P_j} s \in R_i)\}$
      sendtoOtherProcesses(res)
    case $\varphi_1 \wedge \varphi_2$*: res = exchange(*$pSat(K_i, \varphi_1)) \cap exchange(pSat(K_i, \varphi_2))$
    case $EX\varphi$*: res = exchange(*$pred(pSat(K_i, \varphi)))$
    case $EG\varphi : res = fixpoint(pSat(K_i, \varphi), false, Q_i)$
    case $\varphi_1 EU \varphi_2 : res = fixpoint(pSat(K_i, \varphi_1), pSat(K_i, \varphi_2), false)$
  }
  return res
}

2^Q *fixpoint(*2^Q $\varphi, \psi, init$*)*
{
  $Q_{val} = init$
  do
    $Q_{old} = exchange(Q_{val})$
    $Q_{val} = \psi \cup (\varphi \cap pred(Q_{old}))$
  while $Q_{old} \neq exchange(Q_{val})$
  return Q_{val}
}

2^Q *exchange(*2^Q *S)*
{
  res = S
  for each process $P_i \neq pid$
    sendto(P_i*, S)*
  for each process $P_i \neq pid$
    res $= res \wedge receivefrom(P_i)$
  return res
}

process P_i checks the satisfaction of the CTL formula (which is in PNF) using the adapted symbolic algorithm of model checking (Algorithm 4) presented below.

If the satisfaction check can not be decided only after reception of the results of other processes, the process running $pSat_i$ will send messages to receive information from these processes. These steps are repeated until a fix-point is reached (global stabilization occurs), i.e. until no new information can be computed. The processes repeatedly compute information about truth of formulas and send and receive computed information to and from other processes, respectively. This finishes when a fix-point is reached. Then each process extrapolates information, using the fact that the fix-point has been reached. This is performed repeatedly until the information we search for is computed.

For each state and each formula we want to say if its value has already been computed or not. We consider a value for a state and a formula computed if an appropriate value of $receivefrom()$ has already been defined for some process P_i. After reaching the fix-point at the end it can be necessary to resolve undefined values to be able to continue in the computation, i.e. to consider as computed a value for some state and some formula.

The fact that a fix-point has been reached cannot be detected locally. How- ever, by employing an additional communication ($exchange()$) between computers we are able to determine it. Additional communication between processes is also needed to find out what tuples (S, φ) are minimal. Suppose a fix-point has been reached. Each process P_i computes a set of tuples that are minimal in the set for which is undefined. When finished, it sends the set to every other process and receives similar information from other processes. Using this information, each process is able to determine what tuples are minimal in the undefined set.

5.1 Correctness

To prove the correctness of the distributed algorithm, we assume the sequential algorithm is correct. The sequential algorithm evaluates a formula by computing the set of states satisfying this formula. In the distributed algorithm every such set is owned by the process running the root Kripke sub-structure which is also portioned among the processes in a conjunctive way. In the proof we show that, for every CTL formula, the set of states computed by the sequential algorithm is identical to the set computed by the processes in the distributed algorithm.

Theorem 2 `(Correctness)`. *Let* φ *be a CTL formula,* K_i*,* $i = 0, 1, \cdots, n$ *are the fragments of* K *computed by the algorithm 1. If* $Sat(K, \varphi)$ *and* $pSat_i(K_i, \varphi)$ *for* $i = 0, 1, \cdots, n$ *terminate, then* $Sat(K, \varphi) = \bigwedge_{i=0}^{n} pSat_i(K_i, \varphi)$.

Proof. We prove the theorem by induction on the structure of φ. If $\varphi = p$ for $p \in AP$. The sequential algorithm $Sat()$ returns the union of all the states in Q that are marked by p. The process running the root Kripke sub-structure in the distributed algorithm returns

$$res_{id} = (S_{id} \wedge \bigwedge_{j \neq id} receivefrom()) = \bigwedge_{j=0}^{n} S_j.$$

This is based on the fact that each process can wait for information which represents the transition condition, from other process.

For formulas of the form $\varphi = \varphi_1 \wedge \varphi_2$, $pSat_{id}(\varphi)$ first computes $pSat_{id}(\varphi_1) \wedge pSat_{id}(\varphi_2)$. At the end of this computation, the global set is:

$$\bigwedge_{i=0}^{n} (pSat_i(\varphi_1) \wedge pSat_i(\varphi_2)) =$$

$$\bigwedge_{i=0}^{n} pSat_i(\varphi_1) \wedge \bigwedge_{i=0}^{n} pSat_i(\varphi_2).$$

By the induction hypothesis, this is identical to $Sat(\varphi_1) \wedge Sat(\varphi_2)$ which is identical to $Sat(\varphi_1 \wedge \varphi_2)$.

For the formula $EX\varphi$, we compute the pre-image of φ ($pred(\varphi)$).

$$pred_{id}(\varphi) = (s_1 \vee s_2) \wedge \bigwedge_{j \neq id} receivefrom() \mid$$

$$\exists s_1' : s_1 \xrightarrow{P_{j \neq id}} s_1' \wedge \varphi(s_1') \wedge \exists s_2' : s_2 \rightarrow s_2' \wedge \varphi(s_2').$$

We have $s_1 \vee s_2$ is in Q_{id}. Then,

$$pred_{id} = \bigwedge_{j=0}^{n} S_j.$$

If the formula is $EG\varphi$, we would like to prove $\bigwedge_{i=0}^{n} pSat_i(EG\varphi) = Sat(EG\varphi)$. So, we need to prove that $\bigwedge_{i=0}^{n} fixpoint_i(S, false, Q_i) = fixpoint(S, false, Q)$, where S is the satisfaction set of φ. The following lemma proves stronger requirements.

Lemma 3 *Let Q^j be the value of Q_{val} in iteration j of the sequential fixpoint algorithm. Similarly, let Q_{id}^j be the value of Q_{val} in iteration j of the distributed fix-point algorithm in process id. Q^0 is the initialization of the sequential algorithm; Q_{id}^0 is the initialization of the distributed algorithm. Then, for every $j : Q^j = \bigwedge_{i=1}^{k} Q_i^j$. If the sequential $fixpoint$ algorithm terminates after i_0 iterations then so does the distributed $fixpoint$ algorithm.*

Proof. We prove the lemma by induction on the number j of iterations in the loop of the sequential function $fixpoint$. At iteration 0, the initialization of the sequential algorithm, as well as the distributed algorithm is $false$. Hence, $Q^0 = Q_{id}^0 = false$ which implies $Q^0 = \bigwedge_{i=0}^{n} Q_i^0$. Both algorithms perform at least one iteration, so they do not terminate at iteration 0.

Assume Lemma 3 holds for iteration j. We prove it for iteration $j+1$. By the induction hypothesis of Lemma 3 we know that $Q^j = \bigwedge_{i=1}^{k} Q_i^j$. $Q^{j+1} = Sat(Q^j, \varphi)$ and $Q_{id}^{j+1} = pSat_{id}(Q_{id}^j, \varphi)$. Thus, the induction hypothesis of Theorem 2 is applicable and implies that $Sat(Q^j, \varphi) = \bigwedge_{i=0}^{n} pSat_i(Q_i^j, \varphi)$. Hence, $Q^{j+1} = \bigwedge_{i=0}^{n} Q_i^{j+1}$.

The sequential $fixpoint$ procedure terminates at iteration $j+1$ if $Q^j = Q^{j+1}$. We prove that this holds if and only if for every process id, $exchange(Q_{id}^j) = exchange(Q_{id}^{j+1})$. From above, $Q^j = \bigwedge_{i=0}^{n} Q_i^j$ and $Q^{j+1} = \bigwedge_{i=0}^{n} Q_i^{j+1}$.

$$\forall id : exchange(Q_{id}^j) = exchange(Q_{id}^{j+1}) \Leftrightarrow$$

$$\forall id : \bigwedge_{i=0}^{n} Q_i^j = \bigwedge_{i=0}^{n} Q_i^{j+1} \Leftrightarrow$$

$$\forall id : Q^j = Q^{j+1} \Leftrightarrow Q^j = Q^{j+1}.$$

The last equality is implied by the previous one. This completes the proof of the lemma. □
Then the proof of the theorem is completed. □

6 Conclusions and Related Work

In this work we have considered a technique that uses established links between the Kripke sub-structures (fragments) generated by a partitioning algorithm of the state space to perform CTL model checking in a distributed environment. We have developed the necessary theoretical background and described the distributed algorithm. The experimental version of the algorithm is currently being implemented. One of the points that would certainly deserve at least some comments is how to chose the clusters of variables to do partitioning so as to minimize communications. For example we could choose to partition according to few variables that are known to change rarely. We expect to elaborate more possibilities in the future.

In this work in contrast to others, we have focused on the partitioning problem and we have given a complete approach. Other work is based on the modular model checking approach [9] which uses assumptions about missing parts of the state space. An other approach that utilizes a decomposition of the system into parts (modules, fragments) is in [12]. They present a model checking algorithm for pushdown processes and consider the semantics of "fragments" which are interpreted as "incomplete portions" of the process. Another work is the model checking algorithm for the logic EF and CTL and pushdown processes [15]. Finally, in [4] the authors have used 3-valued logic (with ? representing "don't know if property is true or false") to reason about Kripke structures with partial labeling (called partial state space).

For the future work, our first goal is to perform an experimental evaluation. In particular we would like to find

out how the performance is influenced by various types of partition function. We also intend to consider other logics and model checking algorithms in place of the "node algorithm".

References

[1] Barnat J., L. Brim and I. Cerna (2002) Property Driven Distribution of Nested DFS, *VCL'02*, Pittsburgh (USA).

[2] Barnat J., L. Brim and J. Stribrna (2001) Distributed LTL Model-Checking in SPIN, *The 8th International SPIN Workshop on Model Checking of Software (SPIN'01)*, LNCS 2057 Springer-Verlag, pp. 217–234.

[3] Bourahla M. and M. Benmohamed (2002) Predicate Abstraction and Refinement for Model Checking VHDL State Machines, *Electronic Notes in Theoretical Computer Science* **66(2)**, Elsevier Science.

[4] Bruns G. and P. Godefroid (1999) Model checking partial state spaces with 3-valued temporal logics, *11th International Computer Aided Verification Conference*, LNCS Springer-Verlag, pp. 274–287.

[5] Clarke E. M., O. Grumberg and D. A. Peled (1999) *Model Checking*, The MIT Press.

[6] Emerson E., C. Lei (1986) Efficient Model Checking in Fragments of the prepositional μ-calculus, *The First Annual Symposium of Logic in Computer Science*.

[7] Kupferman O. and M. Y. Vardi (2000) An automata-theoretic approach to modular model checking, *ACM Transactions on Programming Languages and Systems* **22**.

[8] Kupferman O. and M. Y. Vardi (1997) Modular model checking, *COMPOS*, pp. 381–401.

[9] Laster K. and O. Grumberg (1998) Modular model checking of software, *TACAS'98*, LNCS 1384 Springer-Verlag, pp. 20–35.

[10] Lerda F. and R. Sisto (1999) Distributed-memory model checking with SPIN, *The 6th International SPIN Workshop on Model Checking of Software (SPIN'99)*, LNCS 1680 Springer-Verlag, pp. 22–39.

[11] Pierre-Alain Masson J. J. and H. Mountassir (2000) Modular verification for a class of PLTL properties, *The 7th International SPIN Workshop on Model Checking of Software (SPIN'00)*,LNCS 1945 Springer-Verlag, pp. 398–419.

[12] Steffen B. and O. Burkart (1994) Pushdown processes: Parallel composition and model checking, *CONCUR'94*, LNCS 836 Springer-Verlag, pp. 98–113.

[13] Tsay Y.-K. (2000) Compositional verification in linear-time temporal logic, *FoSSaCS 2000*, pp. 344–358.

[14] Stern U. and D. L. Dill (1997) Parallelizing the murφ verifier, *Computer Aided Verification (CAV '97)*, LNCS 1254 Springer-Verlag, pp. 256–267.

[15] Walukiewicz I. (2000) Model checking CTL properties of pushdown systems, *Foundations of Software Technology and Theoretical Computer Science*, LNCS 1974 Springer-Verlag, pp. 127–138.

[16] Yorav K. (2000) Exploiting Syntactic Structure for Automatic Verification, Ph.D. thesis, Technion, Haifa, Israel.

On-line Handwriting Chinese Character Analysis and Recognition Using Stroke Correspondence Search

Jungpil Shin
Department of Computer Software, University of Aizu, Aizu-Wakamatsu City, Fukushima, 965-8580, Japan
e-mail: jpshin@u-aizu.ac.jp

Keywords: on-line character recognition, stroke order-free, stroke number-free, stroke information, stroke correspondence search

Received: January 14, 2003

To improve the computational time and recognition accuracy in stroke order- and stroke number-free on-line handwriting character recognition, we have performed a structural analysis of stroke order style and stroke connection. Using handwritten characters chosen from among 2965 Chinese characters, we have deduced information on stroke order and connection using an automatic stroke correspondence system. First, we have shown that the majority of real characters are written in a fixed stroke order, and that the actual stroke order is predominantly the standard stroke order, with about 98.1% of the characters being written in the standard stroke order. Almost all the stroke connections occurred in the standard order (92.8%); while two-stroke connections were common, stroke connections in non-standard order occurred very rarely. In a comparison of our findings with the expected stroke connections, very few were found to actually occur. Second, we have shown methods for incorporating the information in a completely stroke order- and number-free framework. Third, we have shown that the proposed methods can be selected according to the quality of the writer(s), the character(s), and the recognition system. Finally, a large improvement in both computational time and recognition accuracy is demonstrated by experiment.

Povzetek: članek opisuje prepoznavanje kitajskih pisanih znakov.

1 Introduction

On-line recognition of handwritten cursive characters is a key issue in state-of-the-art character recognition research [8], and extensive research has been conducted to mitigate the variation seen in stroke order, stroke number and stroke deformation [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]. In contrast with off-line recognition, on-line recognition has the important advantage of being able to use stroke order and connection information, because the character pattern is expressed by ordered time sequences. Based on stroke order and stroke connection information, there are two approaches to realize on-line character recognition. The first approach actively uses the stroke order and connection information, and the second approach takes into account the inevitable changes in stroke order and connection that occur from person to person, and hence utilizes "*free stroke order*" and "*free stroke number*". Our approach is basically the same as the second method, and the framework for character recognition is stroke order-free and stroke number-free. Using this framework for stroke order-free and number-free character recognition, we can obtain locally available information on the stroke order and connection [15, 16, 17, 18].

The algorithm required to construct a system for stroke order- and number-free recognition needs a correct performance of stroke correspondence between the input pattern and the reference pattern for the highest recognition performance [2, 11, 17, 18]. However, this type of algorithm also carries out a large number of stroke correspondences that do not actually occur. To realize high quality recognition on a small microprocessor with a built-in memory, such as a Personal Digital Assistant (PDA), the recognition time and required computational resources must be reduced. For this purpose, it is expected that a reasonable level of stroke correspondence searching can be realized by using information on the stroke order and stroke connection as they actually occur. The development of a recognition framework that incorporates information on stroke order and stroke connection is expected to achieve a reduction in computational time and to improve the recognition accuracy by neglecting any non-real stroke correspondences.

This paper is based on the assumption that recognition can be performed on the stroke order- and number-free framework. First, we investigate whether useful information can be obtained in regard to the stroke order and connection by focussing on a style analysis of the stroke order variation and the connection between strokes. Second, we propose the methods for incorporating the information in a framework that is completely stroke order-free and stroke number-free. Third, we show that the proposed methods can be selected according to the quality of the writer(s), the character(s), and the recognition system. Finally, a large improvement in both the computational time and the recognition accuracy are demonstrated by experiment.

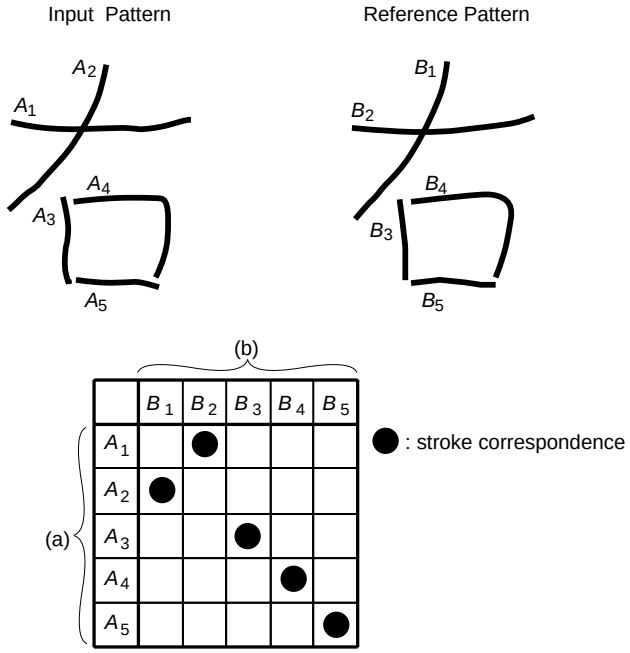


Figure 1: The correct stroke correspondence between: (a) the input pattern, and (b) the reference pattern.

2 The Stroke Correspondence Problem

An on-line input character can be expressed as an ordered series of writing strokes, i.e.,

$$A = A_1 A_2 \cdots A_k \cdots A_N, \tag{1}$$

where the k-th stroke, A_k, is the time sequence representation of the local feature, a_{ik}, of a character, e.g., the x-y coordinates or stroke direction. This is expressed as

$$A_k = a_{1k} a_{2k} \cdots a_{ik} \cdots a_{Ik}, \qquad I = I(k).$$

The reference pattern can be similarly expressed as

$$B = B_1 B_2 \cdots B_l \cdots B_M \tag{2}$$

$$B_l = b_{1l} b_{2l} \cdots b_{jl} \cdots b_{Jl}, \qquad J = J(l).$$

The stroke number of the input pattern is denoted by N, and the stroke number of the reference pattern is denoted by M, with N being equal to M for correct stroke number recognition.

The measure of dissimilarity between the input pattern stroke, A_k, and the reference pattern stroke, B_l, is calculated using the stroke information of the character shape and position, and is denoted by the stroke distance, $\delta(k;l)$. A one-to-one stroke correspondence is defined by the bijection, $\{l(k)\}$, of the stroke number l of the reference pattern from the stroke number k of the input pattern, as shown in Fig.1. The sum of the stroke distances, $\delta(k;l)$, is used as an evaluation standard of the optimum correspondence. Thus, based on the stroke information, the solution of the following minimization problem is considered to give the optimal stroke correspondence in which the minimum value, $D(A,B)$, is chosen as the measure of matching (i.e., the dissimilarity) [11].

$$D(A,B) = \min_{\{l(k)\}} \left[\sum_{k=1}^{N} \delta(k; l(k)) \right] \tag{3}$$

Solving this stroke correspondence determination problem provides a structural analysis of the pattern, and therefore, the dissimilarity, $D(A,B)$, and the stroke correspondence, $\{l(k)\}$, between the patterns are obtained.

Table 1: Occurrence of changes in stroke number.

Stroke-number change	Occurrence number	Ratio (%)
2	13	0.000050
1	662	0.002560
0	172428	0.666767
-1	54100	0.209201
-2	18933	0.073213
-3	7268	0.028105
-4	3202	0.012382
-5	1105	0.004273
-6	517	0.001999
-7	226	0.000874
-8	99	0.000383
-9	29	0.000112
-10	11	0.000043
-11	7	0.000027
-12	1	0.000004
-13	2	0.000008
Total	258603	

3 Preparation for Analysis

The investigation data consisted of 2,965 categories of Chinese characters, i.e., specified characters in the first level of the Japanese Industry Standard (JIS) code set, as written by 90 university students (total = 258,603 characters). The students were directed to write cursively in a normal manner. The data were recorded using a stylus pen on a liquid crystal display (LCD) tablet. The investigation into the results of the change in stroke number are shown in Table 1. This result is in good agreement with [2, 11, 18].

Reference patterns were generated from the training data by storing the average values of the loci of feature points from non-connected strokes that were extracted by rearranging the strokes according to the correct stroke order. One reference pattern for each category was made.

The input character was transformed into a 256 x 256 mesh plane by preprocessing using redundant elimination, smoothing, size normalization and feature point extraction. The x-y coordinates and movement directional vector between one point and the next were extracted as feature information from the character data. The feature information

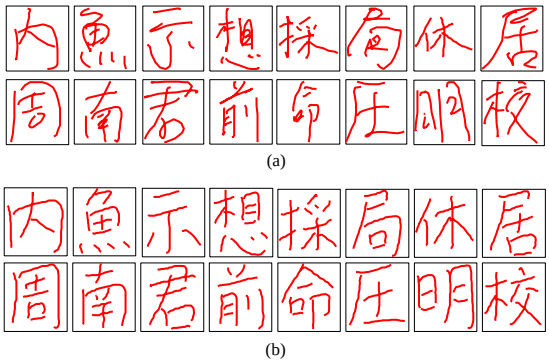


Figure 2: Examples of: (a) the input, and (b) the reference patterns.

of the input and reference patterns was placed into a_{ik} and b_{jl}, respectively. Figure 2 shows typical reference and input patterns.

Automatic searching for the correct stroke correspondence between the input and reference patterns was not easy, since the analysis of the stroke order and the stroke connection had to be performed simultaneously with a high accuracy.

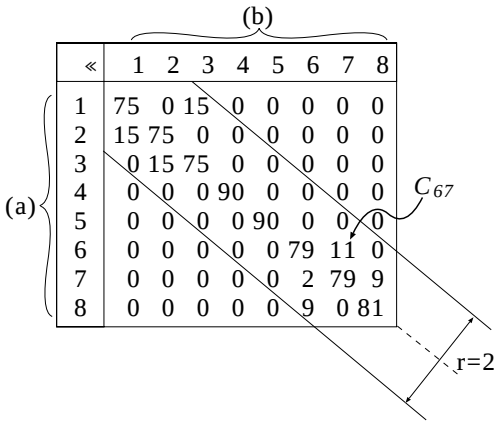
First, the Cube Search method [17, 18] was used to correct the stroke correspondence between the input and reference patterns by automatically searching using backtracking. This algorithm has a novel advantage in being able to search efficiently for the optimal stroke correspondence in spite of: (i) variation in stroke order, (ii) variation in stroke number owing to stroke connections, and (iii) exceptional user-generated stroke deformations. Some of the incorrect stroke correspondences were manually converted into more appropriate stroke correspondences from observation of the characters. These errors arose from strokes written in erroneous positions.

4 Stroke Order and Stroke Connection Information

On the Cube Search method [17, 18], to realize a completely free condition for stroke order, a high percentage of *unreal stroke correspondences* was also carried out.

We expected that a reasonable level of stroke correspondence searching could be realized by using the actual information on the stroke order and stroke connection. Furthermore, there would be no hindrance for the framework of the stroke order- and number-free recognition by use of statistically stable information. The development of a recognition framework that incorporates information on the stroke order and stroke connection is expected to achieve a reduction in computational time, and to improve the recognition accuracy by neglecting non-real stroke correspondences.

We analysed the stroke order and stroke connection of each handwritten character using an automatic stroke correspondence analysis system based on the Cube Search



«	1	2	3	4	5	6	7	8
1	75	0	15	0	0	0	0	0
2	15	75	0	0	0	0	0	0
3	0	15	75	0	0	0	0	0
4	0	0	0	90	0	0	0	0
5	0	0	0	0	90	0	0	0
6	0	0	0	0	0	79	11	0
7	0	0	0	0	0	2	79	9
8	0	0	0	0	0	9	0	81

Figure 3: The occurrence, C_{kl}, of correct stroke correspondence between: (a) the input pattern stroke, k, and (b) the reference pattern stroke, l.

−f	1	2	3	4	5
1	18	72	0	0	0
2	72	18	0	0	0
3	0	0	90	0	0
4	0	0	0	90	0
5	0	0	0	0	90

r=1

^	1	2	3	4	5	6
1	90	0	0	0	0	0
2	0	90	0	0	0	0
3	0	0	90	0	0	0
4	0	0	0	90	0	0
5	0	0	0	0	90	0
6	0	0	0	0	0	90

r=0

‰Ø	1	2	3	4	5	6	7
1	90	0	0	0	0	0	0
2	0	88	2	0	0	0	0
3	0	2	43	21	24	0	0
4	0	0	40	50	0	0	0
5	0	0	5	19	66	0	0
6	0	0	0	0	0	69	21
7	0	0	0	0	0	21	69

r=2

Figure 4: Examples of the occurrence of correct stroke correspondences.

method [17, 18].

4.1 Stroke Order Analysis

Stroke order variation among writers is mainly caused by the particular writing style of the individual. To use the stroke order information in the framework of stroke order-free recognition, changes in the real stroke order data were investigated from the following viewpoints:

(1) the range of the fluctuation of the stroke order in comparison with the standard stroke order, or

(2) whether the stroke order change was completely random.

One such analysed result can be shown using the frequency of an example included in Fig. 3. The quantity C_{kl}

Gap	Examples
0	兄 古 功 欠 旧 幻 犬 弘 元 公 今 勾 孔 午
1	右 左 月 五 号 玄 去 甲 互 止 手 少 火 士
2	花 初 尾 秀 汝 岩 空 居 庁 舌 済 完 升 細
3	辻 何 息 洲 笹 象 我 両 加 柿 報 履 霧 鱈

Figure 5: Character examples, each having a gap.

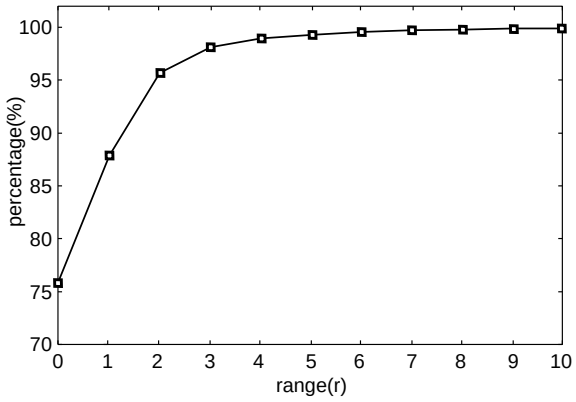


Figure 6: Stroke correspondence distribution.

is the number of the input pattern in which the l-th stroke of the reference pattern is written as the k-th stroke of the input pattern. Note that a connected stroke is composed of strokes that correspond to the reference pattern strokes, based on the fact that the change in stroke order of characters with a connected stroke has the same tendency as the change in stroke order of characters having only a single stroke.

In the example "姓", 15 among the 90 writers sampled wrote the third to the first stroke incorrectly, and the deviations of the 15 incorrect writers are shown in the distribution of the first, second and third strokes. From the results of the stroke order analysis, 76% of all characters were written in accordance with the standard stroke order, and further, the distribution of the remaining characters was concentrated along the diagonal line.

In the occurrence table of Fig. 3, if the maximum distance having a non-zero element from the diagonal line is r, then it is denoted by r-gap. Figure 4 shows examples with an r-gap value. Figure 5 shows character examples, each having a gap. Figure 6 shows the percentage of characters having r-gap, with 95.7% and 98.1% of the characters included in the $r = 2$ and $r = 3$ ranges, respectively.

Based on the above stroke correspondence results, the following methods can be incorporated into the Cube Search algorithm.

(1) If there is no case where the l-th stroke of the reference pattern can be written as the k-th stroke of the input pattern, i.e., $C_{kl} = 0$, then the (k, l) correspondence is excluded.

(2) Based on the tendency that the stroke correspondences are concentrated on the diagonal line, if $|k - l| > r$ then the (k, l) correspondence is excluded.

One of the above methods or the range r can be selected according to the quality of the writer(s), the character(s), and the recognition system. In the example "姓", only 15 pairs of $C_{kl} \neq 0$ or 34 pairs within $r = 2$ were considered as objects suitable for searching for correspondences. By using the stroke order information, a large improvement is expected in terms of the recognition accuracy and computation time.

Connections	Characters	Connections	Characters
(2)→(1)	了 子 孔	(3)→(1)	幼 玄 紙
(2)→(1)	栃 盾 折	(3)→(1)	形 珍 髪
(2)→(1)	民 尼 烏	(3)→(1)	弓 弦 湾
(2)→(1)	見 只 表	(3)→(1)	三 言 書
(2)→(1)	私 去 牟	(3)→(1)	沙 港 準
(2)→(1)	久 残 終	(3)→(1)	立 来 前
(2)→(1)	牛 迄 無	(3)→(1)	口 中 語
(2)→(1)	進 近 導	(3)→(1)	衣 良 裏
(2)→(1)	上 年 雅	(4)→(1)	魚 鳥 燃
(2)→(1)	日 国 網	(4)→(1)	求 函 泰

Figure 7: Typical stroke connections and character examples.

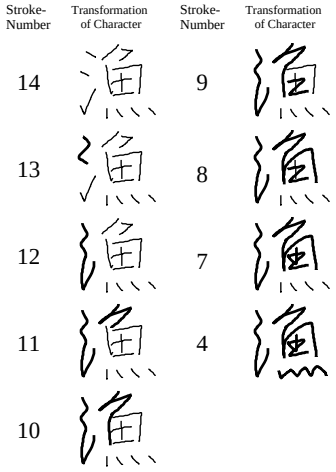


Figure 8: Transformation of the character “ŃŹ” by the occurrence of stroke connections.

4.2 Stroke Connection Analysis

Stroke connection occurs mainly in rapid or in cursive handwriting. As a result, the number of strokes in a character decreases. Figure 7 shows examples of characters in which stroke connections occur, and where some fixed-shape stroke connections occur. Figure 8 shows the transformation of a character from the occurrence of stroke connections. The following points need to be considered for stroke connection.

(1) How many strokes are there in consecutive continuous writing?

(2) In one character, in how many places does independent continuous writing occur?

Table 2: Occurrence of consecutive stroke connection with: (a) standard, and (b) non-standard order.

type of consecutive stroke connection	(a) number	ratio(%)	kinds	ratio(%)
2-strokes	94970	78.88	12779	65.78
3-strokes	15483	12.86	3695	19.02
4-strokes	1093	0.91	751	3.87
5- or more strokes	251	0.21	238	1.23
	(b)			
2-strokes	8036	6.67	1513	7.79
3-strokes	417	0.35	318	1.64
4-strokes	140	0.12	117	0.60
5- or more strokes	15	0.01	15	0.08
total	120405	100	19426	100
single stroke	2580086		32717	

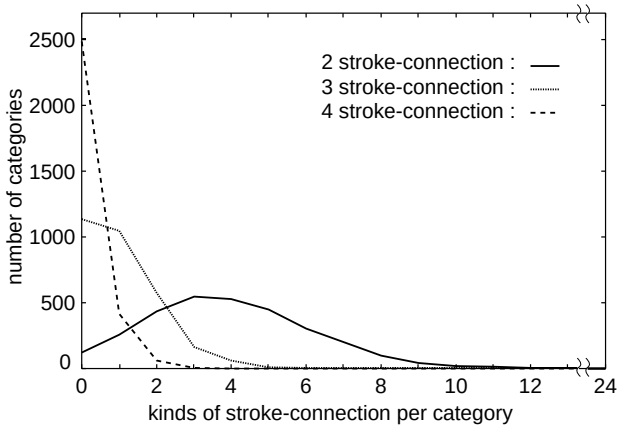


Figure 9: Number of categories having stroke connections.

(3) Are the above occurrences of continuous writing followed by use of a standard stroke order, or by a non-standard stroke order?

The actual situation is complicated. A connection consisting of k strokes is represented as a k-stroke connection.

Table 2 shows the occurrences of consecutive stroke connections, which are divided into standard and non-standard stroke orders. Note that the total number of single strokes is 2,580,086 and includes 32,717 types. Almost all stroke connections occur in the standard order (92.85%); while two-stroke connections occur frequently, stroke connections in non-standard order occur rarely.

The relationship between the connection type and the number of categories is shown in Fig. 9. From these graphs, the connection types 4.82, 1.35 and 0.29 for each character category can be calculated as two-, three-, and four-stroke connections, respectively. As a result, three- and four-stroke connections are 29% and 6% of the two-stroke connections, respectively. Therefore, it can be said that few-stroke connections occur more frequently than many-stroke connections.

Given that a character is composed of N strokes, then $C(N,k)$ (the combination of n objects taken r at a time)

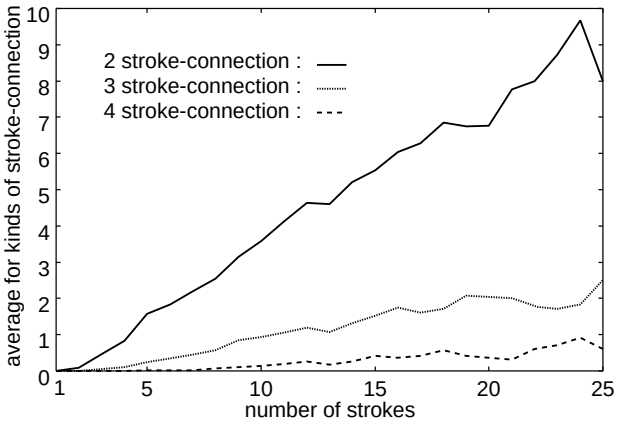


Figure 10: Average of various types of stroke connection.

Table 3: Expected and actual occurrences of each stroke connection type.

stroke-number	expected	actual	actu./exp.(%)
2-stroke connection			
5	$C(5,2)=10$	1.71	17.10
10	$C(10,2)=45$	4.03	8.95
15	$C(15,2)=105$	5.88	5.60
20	$C(20,2)=190$	8.04	4.23
3-stroke connection			
5	$C(5,3)=10$	0.32	3.20
10	$C(10,3)=120$	1.18	0.98
15	$C(15,3)=455$	1.74	0.38
20	$C(20,3)=1140$	2.68	0.235
4-stroke connection			
5	$C(5,4)=5$	0.025	0.500
10	$C(10,4)=210$	0.210	0.001
15	$C(15,4)=1365$	0.456	0.0003
20	$C(20,4)=1995$	0.440	0.0002

connection types of k strokes can be expected. For example, in the case of a character with ten strokes, then $C(10,2) = 45$ two-stroke connections can possibly occur. However, characters have a common stroke order, and most writers compose the characters according to a standard stroke pattern. Figure 10 shows the average stroke connections for each character category. As shown in Figure 10 and Table 3, there were 4.03 two-stroke connections, which was 8.95% of the expected number in a character that consisted of ten strokes. Therefore, many fewer stroke connections than expected occurred in a character.

Based on the above results of stroke connection, the following methods can be incorporated into the Cube Search algorithm.

(1) Using the connection information shown in Table 2, only the stroke connection type that actually occurs is permitted.

(2) In addition to the above condition, a two-stroke connection with standard order is permitted for every possible pair based on the finding that this type occurs

frequently, or

(3) a stroke connection with standard order is permitted for each possible pair, and a stroke connection with a non-standard order is permitted using the connection information.

One of the above methods can be selected according to the quality of the writer(s), the character(s), and the recognition system. By using the stroke connection information, a large improvement is expected in terms of the recognition accuracy and computation time.

5 Experimental Results

The usefulness of the proposed method was demonstrated using PC-based recognition experiments with a Pentium III processor operating at 700 MHz. The training data consisted of 2,965 Chinese character categories, the same as used in the investigation described in Section 3. Reference patterns were generated from the training data by storing the average values of the loci of feature points from strokes, extracted by rearranging them according to their correct stroke order. The test data were provided by 20 writers, and consisted of the same characters as those of the training data (total = 57,667 characters). Twenty categories selected at random were used for the input patterns in groups organized into stroke numbers of 4, 8, 12, 16, and 20.

The input character was transformed into a 128 x 128 mesh plane by using the preprocessing steps of redundant elimination, smoothing, size normalization and feature point extraction. The x-y coordinates and the movement directional vector between one point and the next were extracted as feature information from the character data. The feature information of the input and the reference patterns was placed into a_{ik} and b_{jl}, respectively.

Using DP matching, the stroke distance, $\delta(k;l)$, was calculated using the weighted sum of: (1) the distance of the x-y coordinate sequences, and (2) the distance of the directional vector sequences. An asymmetric DP equation was employed for the DP matching [16].

The evaluation parameter, ρ, of the difference between the inter-stroke information was determined using the distance as

$$\begin{aligned}\rho(k,l;p,q) &= \frac{1}{4}[R(\boldsymbol{d}_{ss}(A_p,A_k),\boldsymbol{d}_{ss}(B_q,B_l)) \\ &\quad +R(\boldsymbol{d}_{se}(A_p,A_k),\boldsymbol{d}_{se}(B_q,B_l)) \\ &\quad +R(\boldsymbol{d}_{es}(A_p,A_k),\boldsymbol{d}_{es}(B_q,B_l)) \\ &\quad +R(\boldsymbol{d}_{ee}(A_p,A_k),\boldsymbol{d}_{ee}(B_q,B_l))]. \quad (4)\end{aligned}$$

where l denotes the stroke number, B_l is the l-th stroke of the reference pattern, $\boldsymbol{d}_{ss}(A_p,A_k)$ is the vector from the start point of A_p to the start point of A_k, $\boldsymbol{d}_{se}(\cdot,\cdot)$ is the vector from a start point to an end point, $\boldsymbol{d}_{es}(\cdot,\cdot)$ is the vector from an end point to a start point, and $\boldsymbol{d}_{ee}(\cdot,\cdot)$ is the vector from an end point to an end point. The weighted sum of the directional difference and the longitudinal difference between vectors is denoted by $R(\cdot,\cdot)$.

Table 4: Recognition experiment results.

	Experiment 1	Experiment 2
Search time(sec)	0.76	0.24
Recognition rate(%)	98.39	98.92

The recognition results were obtained using a force decision that selects the candidate with the minimum dissimilarity, $D(A,B)$. The optimal weighting factors were determined for each experiment.

The following experiments were performed to demonstrate the effectiveness of the proposed method.

(1) Experiment 1: A recognition experiment was carried out performing the stroke correspondence search without using stroke order and connection information.

(2) Experiment 2: A recognition experiment was carried out performing the stroke correspondence search using stroke order and connection information, and where we used the condition (2) of stroke order in section 4.1 and the condition (2) of stroke connection in section 4.2.

Table 4 summarizes the results from the recognition experiments. A large improvement in the recognition rate is achieved by using stroke order and connection information with the stroke correspondence search. The stroke correspondence search time using these data sets can be reduced to about a third of the usual time. The sum total of the recognition time was about 0.65 s, which is the total of the calculation time of δ and ρ and the search times. The reasons for the remaining misrecognitions are: (1) the existence of characters that closely resemble one another, and (2) heavy deformation of a character.

6 Conclusions

In this paper, based on the assumption that recognition can be performed in a stroke order- and stroke number-free framework, useful information with regard to stroke order and connection has been revealed. Using an automatic stroke correspondence system, we have observed the correct stroke order and connection characteristics of 2,965 categories of Chinese handwritten characters. We have shown that the real stroke order of a character mainly follows a standard order, in which the character categories of 95.7% and 98.1% are included within two- and three-strokes, respectively, and 92.85% of all stroke connections are written in standard stroke order. In comparison with the expected stroke connections, very few non-standard connections actually occur. We have described a method for incorporating the information in a framework that is

completely stroke order-free and stroke number-free. Information from either the stroke order and/or the stroke connection can be applied to the framework, which can be altered according to the quality of the data. By introducing this knowledge-based technique into a stroke order- and number-free system, we have achieved a reduction of computational time, and greatly improved recognition accuracy.

Future work will include the development of a technique for efficiently extracting and registering exceptional user-generated stroke deformations. Based on this information, the reference pattern dictionary can be substantially compressed.

References

[1] K. Yoshida and H. Sakoe (1982) Online Handwritten Character Recognition for a Personal Computer System, *IEEE Trans. Consumer Electronics*, vol. 28, no. 3, pp. 202-208.

[2] T. Wakahara, K. Odaka, and M. Umeda (1983) Stroke Number and Order Free On-Line Character Recognition by Selective Stroke Linkage Method, *IECE Trans.*, Japan, vol. J66-D, no. 5, pp. 593-600 (in Japanese).

[3] M. Yurugi, S. Nagata, K. Onuma, and K. Kubota (1985) Online Character Recognition by Hierarchical Analysis Method, *IECE Trans.*, Japan, vol. J68-D, no. 6, pp. 1320-1327 (in Japanese).

[4] K. Odaka, T. Wakahara and I. Masuda (1986) Stroke-Order-Independent On-Line Character Recognition Algorithm and Its Application, *Rev. Electrical Comm. Laboratories*, vol. 34, no.1, pp. 79-85, 1986.

[5] Y. Sato and H. Adachi (1985) Online Recognition of Cursive Writings, *IECE Trans.*, Japan, vol. J68-D, no. 12, pp. 2116-2122 (in Japanese).

[6] K. Ishigaki and T. Morishita (1988) A Top-Down Online Handwritten Character Recognition Method via the Denotation of Variation, *Proc. 1988 Int. Conf. on Computer Processing on Chinese and Oriental Languages*, pp. 141-145.

[7] M. Nakagawa (1990) Non-keyboard Input of Japanese Text – On-Line Recognition of Handwritten Characters as the Most Hopeful Approach, *IPSJ Trans.*, Japan, vol. 13, no. 1, pp. 15-34.

[8] C.C. Tappert, C.Y. Suen, and T. Wakahara (1990) The State of the Art in On-Line Handwriting Recognition, *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 12, no. 8, pp. 787–808.

[9] M. Nakagawa and K. Akiyama (1994) A Linear-time Elastic Matching for Stroke Number Free Recognition of On-Line Handwritten Characters, *Proc. 4th Int. Workshop on Frontiers in Handwriting Recognition*, pp.48-56.

[10] A. Hsieh, K. Fan, and T. Fan (1995) Bipartite Weighted Matching for On-Line Handwritten Chinese Character Recognition, *Pattern Recognition*, Vol.28, No.2, pp.143-151.

[11] T. Wakahara, A. Suzuki, N. Nakajima, S. Miyahara, and K. Odaka (1996) Stroke-Number and Stroke-Order Free On-Line Kanji Character Recognition as One-to-One Stroke Correspondence Problem, *IEICE Trans. Inf. & Syst.*, vol. E79-D, no. 5, pp. 529–534.

[12] Y. Ishii (1986) Stroke Order Free Online Handwritten Kanji Character Recognition Method by Means of Stroke Representative Points, *IECE Trans.*, Japan, vol. J69-D, no. 6, pp. 940–948 (in Japanese).

[13] T. Uchiyama, N. Sonehara, and Y. Tokunaga (1997) On-Line Handwritten Character Recognition Based on Non-Euclidean Distance, *IEICE Trans. Inf. & Syst.*, vol. J80-D-II, No. 10, pp. 2705–2712 (in Japanese).

[14] T. Wakahara and K. Odaka (1997) 'On-Line Cursive Kanji Character Recognition Using Stroke-Based Affine Transformation, *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 19, no. 12, pp. 1381–1385.

[15] H. Sakoe and J. Shin (1995) A Stroke Order Search Algorithm for On-Line Character Recognition, *Technical Report of IEICE*, vol. PRU-95, no. 59, pp. 55–60 (in Japanese).

[16] H. Sakoe and J. Shin (1997) A Stroke Order Search Algorithm for Online Character Recognition, *Research Reports on Information Science and Electrical Engineering of Kyushu University*, vol. 2 no. 1, pp. 99–104 (in Japanese).

[17] J. Shin, M. M. Ali, Y. Katayama, and H. Sakoe (1999) Stroke Order Free On-Line Character Recognition Algorithm Using Inter-Stroke Information, *IEICE Trans. Inf. & Syst.*, vol. J82-D-II, No. 3, pp. 382–389 (in Japanese).

[18] J. Shin and H. Sakoe (1999) Stroke Correspondence Search Method for Stroke-Order and Stroke-Number Free On-Line Character Recognition — Multilayer Cube Search, *IEICE Trans. Inf. & Syst.*, vol. J82-D-II, No. 2, pp. 230–239 (in Japanese).

A System for Evaluation of Human Upper Extremity

Nives Klopčar and Jadran Lenarčič
Jožef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia
nives.klopcar@ijs.si, jadran.lenarcic@ijs.si, http://www2.ijs.si/∼nklopcar/

Keywords: human arm evaluation, reachable workspace, shoulder ranges of motion, rehabilitation

Received: February 13, 2004

A standard method to determine the shoulder range of motion in physiotherapy is to measure the declination angles in predefined planes in the glenohumeral joint. These angles can hardly be understood and visualized and they do not directly interpret the functionality of the arm. In this article, we report a computer program which computes the human arm reachable workspace based on a simplified kinematic model. The program uses as input data the values of the measured declination angles in combination with the proportions of the measured limb. It thus enables to obtain a graphic interpretation of the arm's reachability, quantifies the workspace volume and represents a platform to objectively validate the functionality of the arm. The obtained numerical and graphical results can be used for visualization, computer-aided documentation, comparison between different phases of a rehabilitation process, comparison between different subjects, and can also serve for a deeper mathematical and biomechanical analysis.

Povzetek: članek opisuje sistem za izračun dosegljivega delovnega prostora človeske roke.

1 Introduction

Objective measurements, mathematical processing of the measured data, and effective interpretations and visualizations are of crucial importance in advanced rehabilitation engineering. Many contributions with the purpose to develop new techniques of data capturing, development of new therapeutical approaches and new rehabilitation devices were reported in the literature in the area of human gait. On the contrary, there is a lack of investigations aimed to evaluate the functionality of the human upper extremity. This is primarily because the human gait can be seen as a relatively simple planar motion, while the motion of the human upper extremity is spatial and extremely difficult to evaluate and interpret.

The purpose of this investigation is to develop a computer program which computes, visualizes, and quantifies the human arm reachable workspace (AWS). The human arm reachable workspace is referred to as the volume within which all points can be reached by the chosen reference point on the wrist, namely the center point between *process styloideus ulnae* and *process styloideus radii* [1]. The AWS program is in the process of being introduced as part of a regular measurement and evaluation process of the upper extremity of (in particular hemiplegic) patients in the Institute for Rehabilitation, Ljubljana, Slovenia.

The input data to the AWS program are the arm proportions and the ranges of motion in the joints of the shoulder and of the elbow complexes as measured by the physiotherapist. At the current stage, the measurement technique in the rehabilitation center is entirely manual and only standardized selective motions are measured. The main difficulty, however, is related to the effectiveness of the computation of the workspace from the measured data since this is an extremely time-consuming numerical procedure. To make it useful and implementable on a personal computer, a very concise kinematical model of the human upper extremity must be used in the program. In this investigation, the human arm motion is seen as a combination of the shoulder and the elbow motion. The shoulder motion is composed of elementary motions in the glenohumeral, scapulothoracic, sternoclavicular, and acromioclavicular joint [2, 3, 4, 5]. In order to obtain the reachable workspace effectively, all these motions are modelled as a single spherical joint possessing three perpendicular rotation axes intersecting in the center of the human glenohumeral joint. The elbow joint is understood as a uniaxial joint connecting the ulna with the humerus and the radius with the humerus. These two joints allow the elbow flexion and extension [6] and are modelled as a single rotation. The radioulnar joint, which allows the supination and pronation of the forearm, is not included in the model since it does not influence the spatial position of the wrist.

The workspace is computed as a number of points in space and is then graphically converted into a three-dimensional body by the use of a proper computer graphics module. A physiotherapist can utilize the AWS program to compute and visualize the workspace of the injured arm and compare it with an ideal healthy arm. It is possible to numerically quantify the workspace in terms of its volume, compactness and other mathematical criteria representing the arm functionality.

The first section of this article presents the basic kinematic properties of the human upper extremity and de-

velops a simplified kinematical model. In the following section we discuss the computation of the reachable workspace and its numerical evaluation. In the last section, an example of the treatment of a hemiplegic patient is reported.

2 Simplified kinematics

The movements of the arm are measured relatively to the arm's reference pose, which is when the upper arm is fully extended downward by the side of the body and the forearm flexed for $90°$ (placed forward in a horizontal direction) so that the palm of the hand is turned towards the body. The principal shoulder movements are shown in Figure 1. The shoulder elevation through flexion and retroflexion are measured in the sagittal plane around the coronal axis, the elevation through abduction and adduction is measured in the frontal plane around an anterior-posterior axis. The internal and external rotations are measured in the horizontal plane around a vertical axis [7, 8, 9]. The principal elbow movement is the flexion-extension which is measured in the sagittal plane.

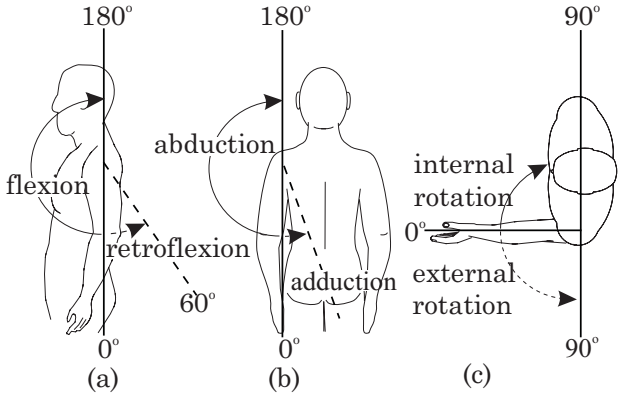


Figure 1: The shoulder complex movements: The elevation through flexion - retroflexion (a), the elevation through abduction - adduction (b) and the internal - external rotation (c).

Figure 2 presents a kinematical model of the arm that replicates the motion characteristics of each single joint included in the arm [5, 6, 10]. Here, U denotes the universal joint that contains two perpendicular rotations, S is the spherical joint that contains three perpendicular rotations, joint T is a translation and joint R a rotation. In the shoulder girdle, the sternoclavicular joint *Sc* is modelled as an universal joint, the acromioclavicular joint *Ac* as a spherical joint, and the scapulathoracic joint *St* as two translations and one rotation. The rest of the shoulder motion is concentrated in the glenohumeral joint *Gh* which is spherical.

The sum of the degrees of freedom in the shoulder girdle is $f = 8$. The number of movable segments is $N = 4$ and the number of joints is $n = 5$. In accordance to the

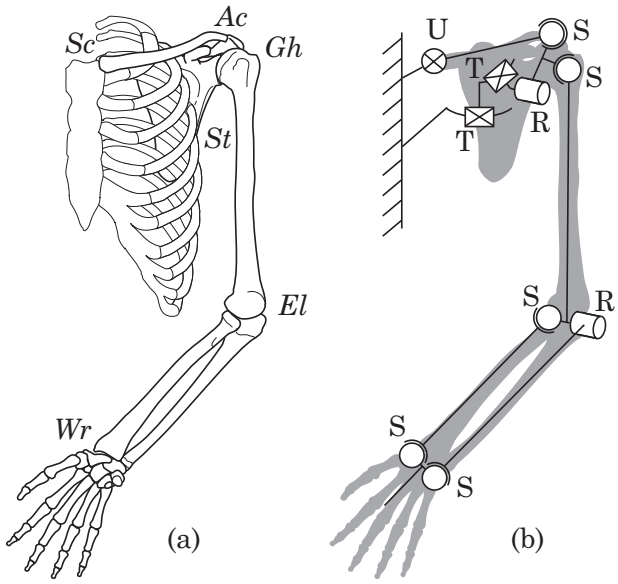


Figure 2: The shoulder complex joints *Sc*, *Ac*, *St*, *Gh*, the elbow *El*, and the wrist *Wr* (a) kinematical model (b).

Grübler's formula only

$$F = \lambda(N - n) + f = 6(4 - 5) + 8 = 2$$

degrees of freedom are independent ($\lambda = 6$ is used since the girdle's motion is spatial). One can see that these two independent degrees of freedom are rotations, meaning that the primary function of the girdle is to point the glenohumeral joint in space. The same motion can be obtained by using only an universal joint U as shown in Figure 3a. If there is a need to also replicate the changing of the size of the girdle, we have to additionally include a dependent translation T which depends on the rotations in the girdle's U joint [11].

In a similar way, we can analyze the kinematical structure of the elbow joint and of the wrist. The ulna is attached to the humerus by a rotational joint R and the radius by a spherical joint S. On the other end, the bones of ulna and radius are attached to the hand by two spherical joints S (Figure 2b). The presented forearm mechanism contains $N = 3$ movable links (the hand is considered as a single rigid link) and $n = 4$ joints. The sum of degrees of freedom in these four joints is f=10. The Grübler's formula gives

$$F = \lambda(N - n) + f = 6(3 - 4) + 10 = 4$$

independent degrees of freedom. They can be replaced by two independent U joints and, eventually, a dependent translation T that models the changing in distance of the forearm (Figure 3a). However, the position of the wrist depends only on the elbow flexion-extension, so that only one degree of freedom (rotation) is needed to model the elbow complex when computing the reachable workspace (Figure 3b). Other three are, therefore, neglected.

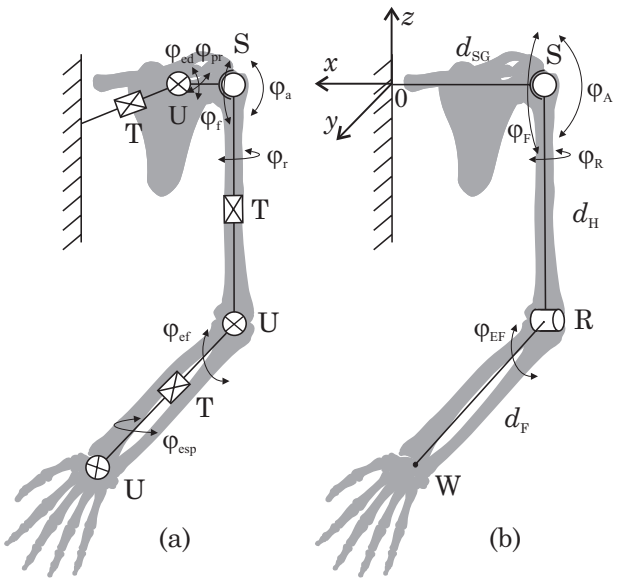


Figure 3: A functional kinematical model (a) and a simplified kinematical model (b) of the arm.

By comparing the joint arrangements in (Figure 3a) with the functional movements of the human arm, it is possible to observe that the first two rotations $\varphi_{\rm ed}$ and $\varphi_{\rm pr}$ represent the elevation-depression and the protraction-retraction of the shoulder girdle with the center in the sternoclavicular joint *Sc*. Angles $\varphi_{\rm a}$, $\varphi_{\rm f}$ and $\varphi_{\rm r}$ represent the humeral abduction-adduction, flexion-retroflexion and internal-external rotations with the rotation center in the glenohumeral joint *Gh*. In order to compute the arm workspace effectively, the movements in the shoulder complex can additionally be simplified by producing the sum of movements in the girdle and in the glenohumeral joints. Thus, the arm elevation through abduction $\varphi_{\rm A}$, elevation through flexion $\varphi_{\rm F}$ and rotation $\varphi_{\rm R}$ are combined by motions in the shoulder girdle and in the glenohumeral joint. For the sake of simplicity it is also assumed that the rotation center is fixed in glenohumeral joint. The final model is seen in Figure 3b.

The complex anatomical properties of the arm do not directly correspond to the presented arrangement of simple rotations about fixed axes [1, 12, 13, 14]. In such a model, therefore, one has to include the interdependencies between the joint coordinates. The lower and the upper limits of joint angles depend on the values of other joint angles. Also the length of the shoulder segment, of the upper arm and of the forearm are dependent on the motions in the simplified joints. Based on the measurements of a number of healthy subjects the correlations between joint coordinates were identified in the past [1].

The simplified model, as presented in Figure 3b, is a very rough approximation of the human arm kinematics. It primarily represents the spatial motion of the reference point W on the wrist. The model includes three rigid segments, the shoulder girdle segment $d_{\rm SG}$ representing the clavicula and scapula, the upper arm segment $d_{\rm H}$ representing the humerus, and the forearm segment $d_{\rm F}$ representing the radius and ulna. We fixed the origin of the reference coordinate inside the body in the region of sterna (Figure 3b). In the reference pose of the arm, when all joint angles are zero, the shoulder segment is parallel to x (medial direction), the upper arm segment to z (vertical direction) and the forearm segment to y (anterior direction). Thus, the segment vectors for the left arm are

$$\mathbf{r}_{\rm SG} = (-d_{\rm SG}, 0, 0)^{\rm T}, \tag{1}$$

$$\mathbf{r}_{\rm H} = (0, 0, -d_{\rm H})^{\rm T}, \tag{2}$$

$$\mathbf{r}_{\rm F} = (0, d_{\rm F}, 0)^{\rm T}. \tag{3}$$

There are three rotations in the glenohumeral joint. The elevation through flexion-extension is expressed as a rotation about axis x. The rotation matrix is

$$\mathbf{R}_{\rm F} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi_{\rm F} & -\sin\varphi_{\rm F} \\ 0 & \sin\varphi_{\rm F} & \cos\varphi_{\rm F} \end{bmatrix}. \tag{4}$$

The elevation through abduction-adduction is a rotation about axis y and the rotation matrix is

$$\mathbf{R}_{\rm A} = \begin{bmatrix} \cos\varphi_{\rm A} & 0 & \sin\varphi_{\rm A} \\ 0 & 1 & 0 \\ -\sin\varphi_{\rm A} & 0 & \cos\varphi_{\rm A} \end{bmatrix}. \tag{5}$$

The internal external rotation is modelled as a rotation about axis z and the corresponding rotation matrix is as follows

$$\mathbf{R}_{\rm R} = \begin{bmatrix} \cos\varphi_{\rm R} & -\sin\varphi_{\rm R} & 0 \\ \sin\varphi_{\rm R} & \cos\varphi_{\rm R} & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{6}$$

It is assumed in this model that the limits of angle $\varphi_{\rm A}$ are constant and independent of other coordinates so that its range is as follows

$$\varphi_{\rm A} = [\varphi_{\rm Am}, \varphi_{\rm AM}], \tag{7}$$

where $\varphi_{\rm Am}$ and $\varphi_{\rm AM}$ are measured in the reference pose of the arm. The limits of the elevation through flexion-extension are linear functions of the elevation through abduction-adduction angle [1]. The range of this coordinate varies as follows

$$\varphi_{\rm F} = [\varphi_{\rm Fm} + \varphi_{\rm A}/3, \varphi_{\rm FM} - \varphi_{\rm A}/6], \tag{8}$$

where $\varphi_{\rm Fm}$ and $\varphi_{\rm FM}$ are measured in the reference pose of the arm. The limits of the internal and external rotation have a quadratic relationship with the elevation through abduction-adduction and flexion-extension angle [1]. Its range varies within the following values

$$\begin{aligned} \varphi_{\rm R} = [&\varphi_{\rm Rm} + 7\varphi_{\rm A}/9 - \varphi_{\rm F}/9 + 2\varphi_{\rm A}\varphi_{\rm F}/810, \\ &\varphi_{\rm RM} + 4\varphi_{\rm A}/9 - 5\varphi_{\rm F}/9 + 5\varphi_{\rm A}\varphi_{\rm F}/810], \end{aligned} \tag{9}$$

where φ_{Rm} and φ_{RM} are measured in the reference pose of the arm. In Equations 8,9 the angles are expressed in degrees.

The elbow flexion-extension rotation is calculated as a rotation about axis x and the corresponding rotation matrix is

$$\mathbf{R}_{\mathrm{EF}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi_{\mathrm{EF}} & -\sin\varphi_{\mathrm{EF}} \\ 0 & \sin\varphi_{\mathrm{EF}} & \cos\varphi_{\mathrm{EF}} \end{bmatrix}. \quad (10)$$

It is assumed that the limits of this angle are constant and independent on other coordinates so that its range is as follows

$$\varphi_{\mathrm{EF}} = [\varphi_{\mathrm{EFm}}, \varphi_{\mathrm{EFM}}]. \quad (11)$$

Here, $\varphi_{\mathrm{EFm}} = -90^\circ$ and $\varphi_{\mathrm{EFM}} = 60^\circ$ are fixed as for the healthy elbow [1, 7, 8].

The position of the reference point W is calculated by

$$\mathbf{r}_{\mathrm{W}} = \mathbf{r}_{\mathrm{SG}} + \mathbf{R}_{\mathrm{R}} \cdot \mathbf{R}_{\mathrm{A}} \cdot \mathbf{R}_{\mathrm{F}} \cdot \mathbf{r}_{\mathrm{H}} + \mathbf{R}_{\mathrm{EF}} \cdot \mathbf{r}_{\mathrm{F}}. \quad (12)$$

3 Workspace computation

The input data to the AWS program are the joint limits measured in the reference pose of the arm, which are φ_{Am}, φ_{Fm}, φ_{Rm}, φ_{EFm}, φ_{AM}, φ_{FM}, φ_{RM}, and φ_{EFM}. These parameters vary considerably among individuals as affected by age, sex, injuries, and stage of the illness [7]. Relatively to the hight of the subject H, the segments of the arm are normalized in accordance to the anthropometric table [15]. The length of the shoulder girdle length is then $d_{SG} = 0,129 \cdot H$, of the humerus is $d_H = 0,185 \cdot H$, and of the forearm $d_F = 0,146 \cdot H$.

The procedure to determine the workspace has few stages. The first stage is to compute the set of points which can be reached by the wrist. This computation involves four nested loops, each associated with one joint angle. In every iteration, position $\mathbf{r}_{\mathrm{W}}$ (Equation 12) is computed and stored as a three dimensional vector. The procedure is repeated until all ranges of joint angles are swept [1]. In addition, the collisions between the segments of the arm (humerus and forearm) and the body (head, neck and trunk) have to be taken into account. If during the computation an arm segment intersects the body, the related position of the wrist is eliminated as impossible and is not considered as part of the workspace. An elliptical cylinder to approximate the body whose size is $0,174 \cdot H$ in the frontal plane and $0,089 \cdot H$ in the sagittal plane is used. The head is approximated by a sphere whose radius is $0,065 \cdot H$ [15].

The resolution is set to 5° for all joint angles. It corresponds to the measurement error in the input data. Since the ranges of coordinates change from one subject to another and throughout the workspace, it is impossible to exactly predict the number of iterations. Usually, tens of thousands of iterations are needed to determine the whole set of points approximating the reachable workspace.

The obtained set of points lies inside a cube of edge $L = 2(d_{\mathrm{H}} + d_{\mathrm{F}})$ whose center is in the center of the shoulder joint. This cube is seen as a volume of n^3 smaller cubes with edge L/n, where n is a desired resolution, which is limited by

$$L/n > (d_{\mathrm{H}} + d_{\mathrm{F}}) \tan 5^\circ. \quad (13)$$

The cubes that do not contain at least one point $\mathbf{r}_{\mathrm{W}}$ are eliminated. As a result, the workspace of the arm is described by a set of cubes of edge L/n.

In order to increase the accuracy, the cubes forming the surface of the workspace are broken into smaller ones with edge of half length. In every step i the workspace volume is computed by

$$V^i = V_{\mathrm{I}}^i + \frac{V_{\mathrm{S}}^i}{2}, \quad (14)$$

where V_{S} is the volume of cubes on the surface, and V_{I} the volume of all other (inner) cubes. The procedure is ended when

$$\nu^i = \frac{|V^i - V^{i-1}|}{V^{i-1}} \quad (15)$$

is smaller than a prescribed value ν.

The last stage in determining the arm workspace is to smoothing the surface cubes. For this purpose a Bezier interpolation [16] is performed. The workspace can thus be visualized with different color textures, illuminated with different positions of light, or made transparent.

A comparison between the calculated and the measured reachable workspace of the healthy left arm is shown in Figure 4. The workspace in Figure 4a was measured in equidistant planes in [1]. Such a measurement is extremely complicated and time consuming. It can only be performed on healthy subjects. On the contrary, the calculated workspace (Figure 4b) can be obtained based on very simple manual measurements which are part of a standard procedure in treating hemiplegic patients in the rehabilitation center. The similarity between the measured and the calculated workspace is quite evident.

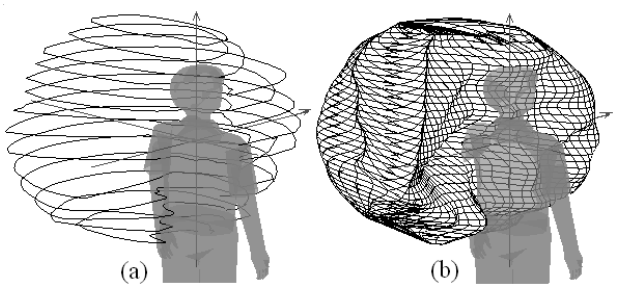


Figure 4: A comparison between the measured (a) and the calculated (b) reachable workspace.

4 Example

An example of a treatment of a hemiplegic patient is presented in Figure 5. The figure shows the reachable workspace of the left handicapped arm before the treatment

(W_{L1}) and two different phases during the treatment (W_{L2} and W_{L3}). Workspace W_{R0} is associated with the healthy right arm inverted to the left hand side for comparison. The measured ranges of motion are collected in Table 1.

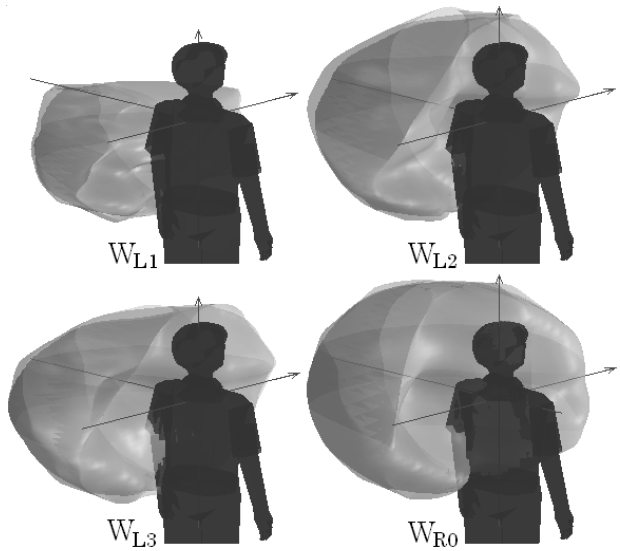


Figure 5: A comparison of the reachable workspace of the left handicapped arm with the reachable workspace of the right healthy arm of a hemiplegic patient.

The measured ranges of motion of the handicapped left shoulder before and during the treatment, as well of the healthy right shoulder, are reported in Table 1. The elbow flexion-extension angles are taken as for the healthy arm and are $\varphi_{\mathrm{EFm}} = -90°$ and $\varphi_{\mathrm{EFM}} = 60°$.

Table 1: The measured shoulder ranges

	φ_{Fm} φ_{FM}	φ_{Am} φ_{AM}	φ_{Rm} φ_{RM}
W_{L1}	$-45°$ $65°$	$-10°$ $80°$	$-55°$ $25°$
W_{L2}	$-55°$ $140°$	$-10°$ $95°$	$-60°$ $40°$
W_{L3}	$-80°$ $120°$	$-10°$ $95°$	$-45°$ $50°$
W_{R0}	$-60°$ $170°$	$-10°$ $170°$	$-60°$ $90°$

Table 2 shows the computed volume of the reachable workspace of the handicapped left arm and of the healthy right arm. The workspace was computed with $\nu \leq 10\%$ relative error. The hight of the subject is $180,0$ cm.

Note the workspace volume is not directly associated with functionality of the arm. Other indices in combination with the volume can be used, such as the workspace compactness [17]. The workspace compactness quantifies the similarity of the workspace shape with a sphere. It is assumed that a compact workspace is more adaptable then an elongated one. Also the location of the workspace relative to the body is an important issue. A reachable space in front of the body could in general be more useful.

Table 2: The computed workspace volume for $H = 18,00$ dm

	volume $\pm\nu$
W_{L1}	$190,4$ dm^3
W_{L2}	$401,4$ dm^3
W_{L3}	$364,3$ dm^3
W_{R0}	$598,2$ dm^3

The AWS program is written in Matlab and converted into an autonomic form. It is designed to be used at the Institute for Rehabilitation as a standard tool for the examination of the shoulder complex. The input data form contains an identical information as the paper form used in the past. The workspace block is added (Figure 6).

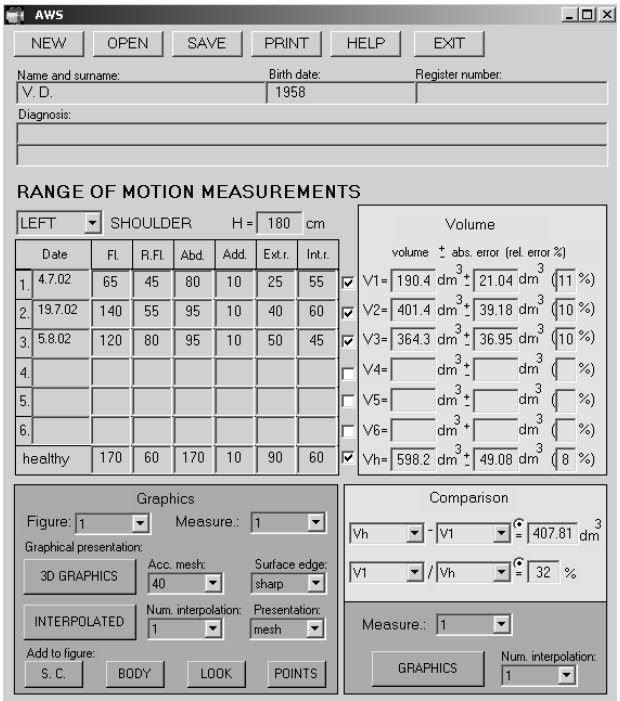


Figure 6: The input form of the AWS program.

The workspace is visualized in a posterior and in an anterior view. It is possible to superimpose the workspace of the handicapped arm onto the one of the healthy arm by using a transparent envelope (Figure 7). Thus, the workspaces can be directly compared during different phases of the rehabilitation process. The results can easily be documented and stored in a computer or printed. They can be displayed and numerically processed, as well

as electronically transferred to another user.

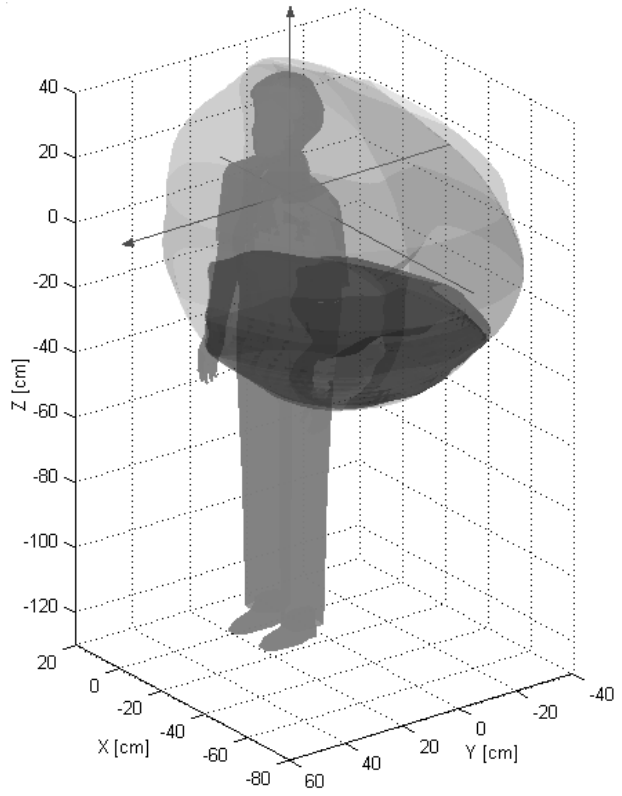


Figure 7: A direct comparison of the workspace of the handicapped arm (W_{L1}) with the one of the healthy arm (W_{R0}) by superimposing the two workspaces.

5 Conclusion

A computer program which computes the human arm reachable workspace is reported in this article. The program is based on a simplified kinematic model of the human upper extremity in which the shoulder complex is approximated by a spherical joint and the elbow complex by a rotation. The input data to this program are taken from a standard evaluation procedure in physiotherapy. The reachable workspace can be quantified by its volume or other mathematical indices.

The main advantage of this program is that it can visualize the reachability of the measured human arm. The obtained results can be used for computer-aided documentation, numerical or visual comparison between different phases of a rehabilitation process, and numerical or visual comparison between different subjects. This helps us to plan and control the rehabilitation procedure of shoulder patients. The program is now being introduced in the Institute for Rehabilitation, Ljubljana, Slovenia.

Acknowledgement

This investigation was supported by the Slovenian Ministry of Education, Science and Sport. We are grateful to the employees of the Institute for Rehabilitation, Ljubljana, Slovenia, for their valuable contribution.

References

[1] J. Lenarčič, A. Umek (1994) Simple model of human arm reachable workspace, *IEEE Trans. System, Man and Cybernetics*, Vol. 24, pp. 1239–1246.

[2] V. M. Zatsiorsky (1998) *Kinematics of human motion*, Human Kinematics.

[3] A. E. Engin and S. T. Tumer (1989) Three-dimensional kinematic modeling of the human shoulder complex - part I: Physical model and determination of joint sinus cones, *Trans. of the ASME Jour. of Biomech. Eng.*, Vol. 111, pp. 107–112.

[4] J. E. Wood, S. G. Meek and S. C. Jacobsen (1989) Quantitation of human shoulder anatomy for prosthetic arm control - II. Anatomy Matrices, *Jour. of Biomech.*, Vol. 22, No. 4, pp. 309–325.

[5] Z. Dvir and N. Berme (1987) The shoulder complex in elevation on the arm: a mechanism approach, *Jour. of Biomech.*, Vol. 11, pp. 219–225.

[6] J. Lenarčič and M. Stanišić (2003) A humanoid shoulder complex and the humeral pointing kinematics, *IEEE Trans. on Robotics and Automat.*, Vol. 19, No. 3, pp. 499–506.

[7] C. C. Norkin and D. J. White (1985) *Measurement of joint motion: A guide to gonimetry*, F. A. Davis Company Philadelphia.

[8] I. A. Kapadanji (1970) *Physiology of the Joints*, Churchill Livingstone, London.

[9] D. J. Magee (1997) *Orthopaedic Physical Assessment*, W. B. Saunders Company, 3rd ed.

[10] J. Hesselbach, M. B. Helm, H. Kerle, M. Frindt and A. M. Weinberg (1998) *Advances in Robot Kinematics: Analysis and Control*, Kluwer Academic Publishers, pp. 551–560.

[11] J. Lenarčič, M. M. Stanišić. V. Parrenti-Castelli (2000) Kinematic design of a humanoid robotic shoulder complex, *Proc. Int. Conf. On Robotics and Automat.*, San Francisco, USA.

[12] V. T. Inman, J. B. Saunders, L. C. Abbott (1944) Observation on the function of the shoulder joint, *Jour. of Bone and Joint Surgery*, Vol. 26, pp. 1–30.

[13] C. Högfors, B. Peterson, G. Sigholm and P. Herberts, (1991) Biomechanical model of the human shoulder joint - II. The shoulder rhythm, *Jour. of Biomech.*, Vol. 24, No. 8, pp. 699–709.

[14] S. D. Bagg and W. J. Forrest, (1980) A Biomechanical analysis of scapular rotation during arm abduction in scapular plane, *American Jour. of Phy. Med. Rehabilitation*, Vol. 67, No. 6, pp. 238–245, 1980.

[15] D. A. Winter (1990) *Bimechanics and motor control of human movement*, Wiley-Interscience Publication, University of Waterloo.

[16] F. Gerald (1990) *Curves and surfaces for computer aided geometric design*, Academic Press Inc. - Harcourt Brace Jovanovich Publisher.

[17] J. Lenarčič (1992) An approach to optimum design of robot manipulators, *Laboratory Robotics and Automation*, Vol. 4, pp. 137–143.

A Software Architecture for Enterprise Components

D. A. Helton
Computer Information Systems
Northern Michigan University
Marquette, Michigan, USA
E-mail: dhelton@nmu.edu

Keywords: component-based software development, enterprise components, software components, component software, COTS

Received: January 31, 2004

Component-based software development, the idea of constructing computer programs from existing software modules, has attracted considerable attention. Component size has become a significant consideration. Recently, there have emerged significant research prototypes and early commercial systems for constructing major business applications from a few components of coarse granularity, termed enterprise components, rather than combining hundreds of small components. This research with large components has shown that building applications from components of coarse granularity presents distinct challenges from assembling systems from small elements. The focus of this paper is a proposed system architecture, synthesizing the best features of other systems using enterprise software components, and adding new features, as needed, as a step towards the improvement of software development involving large components.

Povzetek: članek opisuje gradnjo poslovnih sistemov iz velikih modulov.

1 Introduction

There is growing interest in coarse-grained software components, or *enterprise components* (Levi & Arsanjani 2002), which offer the potential of building applications quickly from a few large modules rather than by composing numerous fine-grained components. An analysis of an organization's functional areas determines what is to be included within each of these large components. Each major process becomes a single enterprise component (Bandini et al., 2002, Helton 1999, Levi & Arsanjani 2002).

The objective of this study is to derive an improved system architecture for software development with large components, which will be presented in sections 2, 3 and 4 of this paper, as follows: (section 2) a review and synthesis of features from existing systems, (section 3) the definition of functionalities of system components and interrelationships among them, and (section 4) the system architecture specification.

2 Review and Synthesis of Previous Systems

This stage includes three activities: (1) review existing systems handling large components to determine requirements for an improved software architecture, (2) locate candidate features that might fulfill these requirements within these systems or other sources, (3) select a group of items from the candidates for synthesis into an improved system. The first and second activities were completed in a previous study (Helton 1999) and a summary of the results will be included below. This preliminary study contained an examination of research prototypes and early commercial systems using enterprise components, including Carnegie-Mellon University Software Engineering Institute (SEI) reports of development with commercial-off-the-shelf (COTS) systems (Carney 1997), work at Harvard University's Brigham & Women's Hospital on large component imaging software (Deibel & Greenes 1995), Sun Microsystems' Enterprise JavaBeans (EJB) model (Thomas 1997), the large component version of SAP *enterprise resource planning (ERP)* applications (SAP 1997), and Stanford University's CHAIMS project on very large-grained, heterogeneous, distributed components (Beringer et al. 1998). The study (Kim 2002) of the Korean government's initiative in promoting enterprise component development also was beneficial.

Review of Other Systems. As an immature technology, existing large-component software systems do not meet adequately the needs envisioned by their designers. The aforementioned study (Helton 1999) of the other large component systems uncovered some of their deficiencies. Despite encountering these inadequacies, the study of these and other related systems provides a source for locating functionalities that meet these requirements. Theoretically, building systems from a few large components should be less challenging than building a system from many small components. The study of existing systems indicated the following needs for enhanced large-components systems: (1) streamlined components, (2) simple interfaces, (3) straightforward composition, (4) plug-and-play compatibility, and (5) standard distribution infrastructure.

The aforesaid review of these systems (Helton 1999) suggests a need for simplifying rather than complicating the large components. Several of the large component systems described do not subordinate appropriately the details of coarse-grained components. Scrutiny of these systems indicates that large components manifest a complexity that makes their composition with other elements significantly more challenging than assembling systems from fine-grained components.

The previous study (Helton 1999) also indicated that, as the granularity moves up the scale, the appropriateness of using the conventional set of object-oriented (OO) features becomes suspect. Several of the systems (Enterprise JavaBeans, SAP R/3, and the Harvard information system research) overdo object orientation by extending it to the highest level, i.e., to that of the overall large component structure. The design of the inner structure or implementation of a large component, which is not the focus of this paper, may follow extensively the OO paradigm, if the designer so chooses. Even in working on the implementation design, great care must be taken to avoid creating mazes of confusing OO inheritance hierarchies. Nevertheless, the basic concept of OO, that is, depicting in the software an entity from some business domain, was appropriate for each system studied.

The preliminary study (Helton 1999) also revealed that the use of OO features beyond this inner structure presented problems in these other systems. Since only a few components are involved, many elements of OO are superfluous. The inherent complexity of large components makes them challenging enough to handle without adding further complications. Large components are stable and do not require dynamic features for their quick replication or frequent modification. As mentioned earlier, each enterprise component might represent a distinct, complex business function, such as a payroll system or an inventory process. Instantiation, when only a few components of this granularity are involved, is a needless complication. As observed by Taylor (1995), the reusability of components decreases as complexity increases. At this point of component granularity, reusability is minimal. Thus, the small reusability attained by permitting inheritance is overwhelmed by the disadvantages generated by entangling class hierarchies. Following the same logic with regards to the diminishing returns of reusability as components increase in size, other OO features offer little in return for the baffling complexity required to implement them.

The previous review (Helton 1999) of the systems also shows that simple interfaces are another essential requirement to facilitate the handling of large components. The systems mentioned above further complicate large component systems by using OO inheritance for interfaces of large components. This overuse of OO inheritance can permit direct access to deeply nested interfaces within a large component (D'Souza & Wills 1998). This contradicts the basic idea of hiding component details behind interfaces (Szyperski 1997).

Streamlining component interfaces may be achieved by delegation (Lewandowski 1998). The overall component contains a straightforward interface, providing the only internal implementation access to external clients. The large component may contain nested interfaces, even several layers deep. Access to these inner interfaces is via forwarding or delegation, which restricts flexibility to some degree, but keeps component structure simple.

The examination of other large component systems (Helton 1999) brought to light another requirement for improving this type of system. In order to work effectively, the system needs a straightforward means of connecting together the enterprise components. If the cumbersome OO interface and implementation hierarchies are eliminated, then an uncomplicated glue component may be inserted to tie pre-existing applications together. The Stanford CHAIMS researchers used this technique, and D'Souza and Wills (1998) also suggested it.

Plug-and-Play compatibility of large business components from multiple vendors is another major requirement revealed by the earlier study (Helton 1999) of these systems. Though the ERP vendors allow competitors to add secondary modules, none of these vendors guarantee that their major modules may be substituted by comparable components produced by competitors. Experience with component projects has indicated that effective reuse of large components is generally restricted to vertical industry segments (Szyperski 1997). This conforms to the reusability principle, where the more specialized a module is, the less reuse potential it has. Thus, the feasibility of using components across industry boundaries is much more restricted for large components than it is for small ones. The Harvard information systems project confirms the need for establishing standards within particular business domains (Deibel & Greenes 1995).

The interchangeability of components from multiple sources requires, foremost, standardization of the specification, i.e., the interface and related descriptions about the functionality of the enterprise component. Uniform implementations are not required for plug-and-play compatibility of components. Researchers at PeopleSoft and SAP also confirm the need to develop specialized, standard interfaces for particular industries. Experience has demonstrated that standardizing interfaces for specific industry groups yields adequate compatibility to effectively assemble the respective large components.

A standard distribution infrastructure is one of the most important requirements for development with large components. Companies usually expect that new major applications will be deployed over networks. The

pressures of competition have produced incompatible distribution systems (Szyperski 1997), making development with large components challenging. Some high-level data models abstract the complexity of data storage mechanisms by simply depicting persistent data as being contained within application modules. The conceptual framework (Helton 1999) for the proposed architecture, however, incorporates, at the top level, software elements of large granularity connected by a distribution system, as well as a data storage facility, normally a database management system (DBMS).

Originally, CORBA, DCOM, and Java, were not designed to support large software components (Szyperski 1997). Enterprise JavaBeans extends Java to the point that EJB is a whole, distinct component system, rather than just a distribution system. Though capable of handling large components, EJB is an adaptation of a system designed to handle small software elements. EJB thus retains the overuse of some OO features that is commonplace in systems focusing upon small components. Those working with large component development described in the aforesaid study (Helton 1999), pertinent to COTS, Harvard University's Brigham & Women's Hospital, EJB, ERP, and CHAIMS projects, as well as the recent initiative in Korea (Kim 2002) found that it was difficult to find existing infrastructures that supported their conceptual models for handling large-grained components. For example, in the Harvard information systems project (Deibel & Greenes 1995), investigators initially developed an attractive architecture for handling large components. As the work progressed, however, the Harvard researchers realized that no existing distribution systems completely supported their model. Thus, they modified their architecture to conform to the requirements of existing infrastructures, oriented towards components of small granularity.

The complexity and magnitude of building business applications requires developers and researchers to focus upon assembling large components, despite the inadequacies of underlying distribution systems that are available. The preliminary study (Helton 1999) of early commercial and research prototypes of large component systems have been forced to use infrastructure systems that do not adequately support their coarse-grained components. The Stanford CHAIMS was unique in that, rather than compromising by using an incompatible infrastructure, the researchers developed their own infrastructure (Beringer et al. 1998).

Thus, this concludes the review of existing systems handling large components in order to establish requirements for improving large-component systems. As noted earlier, these requirements are: (1) streamlined components, (2) simple interfaces, (3) straightforward composition, (4) plug-and-play compatibility, and (5) standard distribution infrastructure. The next activity is to locate feasible elements that satisfy these five requirements.

Locate Candidate Features. The review of prior systems, in the foregoing section, suggests that the requirements for a large-component architecture may be met by making the following enhancements: (1) acceptance of attractive features used with the structured approach to software development, (2) removal of inappropriate object-oriented features, (3) patterning each enterprise after a complete business process, (4) adoption of solid notions from the client/server model, and (5) employment of standards specially designed for the large components.

In developing the conceptual framework for the architecture (Helton 1999), candidate features were obtained from the following sources: (1) structured programming, (2) object orientation, (3) process-oriented notions, (4) the client/server model, and (5) standards concepts. In this step, locate candidate features, the four preceding sources of features for an enhanced software architecture will be examined further.

High cohesion and loose coupling, two concepts associated with structured development, enhance component development (Budgen 2003, Sametinger 1997). Inclusion of these two features with the suggested architecture is possible because unattractive inheritance hierarchies and other inappropriate OO notions have been expunged. Insuring that everything pertaining to a particular function adheres to a central, unifying purpose attains high cohesion. Loose coupling is achieved by eliminating all connections between components other than data linkages.

Structured interfaces, derived from structured programming, should be beneficial to the proposed architecture. Several systems (Enterprise JavaBeans, Harvard health information systems, SAP R/3), unnecessarily complicate development with large components by including OO interfaces, which produce puzzling interface hierarchies. The rationale for including this maze of interfaces is to permit direct external access to smaller components embedded within larger ones. Even if *component* inheritance and other inappropriate OO features are eliminated, this compromise, consisting of the inclusion of *interface* inheritance, potentially degrades the effectiveness of a system. The Stanford CHAIMS project confirms that procedural (structured) interfaces streamline development involving large components. Though these structured interfaces only allow indirect communication with embedded components, they enhance the construction of applications with large components by subordinating minutia.

Information hiding (Inverardi & Tivoli 2002, Ravichandran & Rothenberger 2002) is another feature from structured programming suggested for the proposed architecture. Information hiding abstracts the intricacy of a large component's inner workings. Even a baffling maze of OO inheritance hierarchies, for example, may be concealed through this feature. A large business

component, comprised of an aggregation of smaller components, should limit access to nested elements via delegation instead of exposing inner interfaces (Lewandowski 1998).

Component development with small components incorporates many features that do not function well with enterprise components. Several of the systems (Enterprise JavaBeans, SAP R/3, Harvard health information systems research), however, overextend OO concepts to large components, making their handling unduly complex. The central OO notion of mapping business entities directly to software is worthwhile for large components. A modified version of OO *encapsulation* is useful for large components, too. Simply expressed, this means that an object *class* encapsulates both data and related operations on the data (Pressman 1997). The implementation of a component may follow any model desired (Buck-Emden 1996, Short 1997, Szyperski 1997). If the inner structure follows the OO model, a single enterprise component may have many object classes. A large component represents a function from the business domain and encapsulates everything pertaining to this function (D'Souza & Wills 1998, Levi & Arsanjani 2002, Shaw et al. 1995). A component may even subordinate, through abstraction, all of the mechanics involved with attaining persistence of data, that is, storage of data, generally in a DBMS.

Most other OO concepts are neither necessary nor beneficial for handling large components. Enterprise components tend to be more static in nature than small components (D'Souza & Wills 1998). The implementation of a large component may or may not follow OO structure. The details of the inner structure remain hidden and consequently are insignificant at the high level view of the architecture. This is consistent with the concept of black-box assembly of components (Inverardi & Tivoli 2002, Ravichandran & Rothenberger 2002). In composing large components, the developer works at a level of abstraction where the concern is upon the connections between components. Thus, the study of other systems (Helton 1999) made it clear that at this high level, components need to be simplified, irrespective of the complexity of their inner workings.

Extending OO ideas, other than the aforementioned mapping of business functions to software entities, is detrimental development with large components, as illustrated by the CHAIMS project at Stanford University (Beringer et al. 1998). Confusion exists, among many developers, about the differences between OO and component approaches to development (D'Souza & Wills 1998). Development with coarse-grained components should capitalize on the static demeanor of these large components by reducing their number to just a handful. Having classes instantiate enterprise components only complicates development with large components and is unnecessary. Since only a few large, stable components are involved in these systems, using OO inheritance, and thereby possibly creating entangled dependency hierarchies, is even more inappropriate. Multiple inheritance has the potential of creating extremely confusing class hierarchy webs (Hissam et al., 2002, Pressman 1997) and thus is controversial even when working with small grained components, and is highly questionable for composing large components. In addition, removal of inheritance eliminates problems with related features, particularly those associated with overriding and polymorphism. Since large components are inherently static, and thus use static binding, late binding is generally unnecessary.

Though mapping elements from the business domain to software is a central OO notion, the process-oriented manifestation of this idea is better suited to large component development. This means that an entire business process may be portrayed within an application as a large component (D'Souza & Wills 1998, Levi & Arsanjani 2002).

At two levels, client/server concepts may be included in the large component architecture (Helton 1999). First is the common practice of transforming a program into modules, with a central client module calling server elements. The business components communicate only through a coordinating glue component. At the second level, the client/server model may be employed in the architecture to demonstrate the interaction of concurrently executing processes. While the physical distribution of these processes was implicit in the conceptual framework (Helton 1999), this feature will be specified explicitly in the proposed architecture.

The review of the coarse-grained systems (Helton 1999) suggests several sources suggests that standards are needed for: (1) large components, in general, to provide functionality identical to that of equivalent units available from other vendors; (2) interfaces, to make components plug compatible with those from other companies; and (3) underlying distribution systems, so that developers may focus upon the application software and so that the distribution infrastructure is able to handle large components appropriately. Though at times an elusive goal, significant precedents for de facto or formal standards exist in the software field (Szyperski 1997) and meaningful component development requires standards (Hissam et al. 2002).

A standard for large components would include specifications to insure that the behavior of a component from one vendor is equivalent to a comparable component from a competitor (Vitharana 2003). Components with standard, that is identical, interfaces would be plug compatible with each other. Both of the foregoing standards would be necessary. Otherwise, one might expect components with identical interfaces but dissimilar specifications to function correctly. They would, however, yield erroneous functionality. Standard distribution infrastructures are absolutely essential since communication between elements is impossible without using the same distribution system. Components must be

compatible in each of these three areas in order to interact properly. Consequently, a particular ERP vendor, for example, must maintain rigid internal standards within the corporation in order to attain component functionality. The only other possibility would be to rig heterogeneous elements together with wrappers and other patches. As discussed in reviewing these systems, vendors have been reluctant to enter into agreements for standards that would make components interchangeable between companies.

ERP vendors do not provide general, straightforward plug compatibility (Helton 1999). These companies include mechanisms for appending peripheral components, but protect their own key business components so that they are not interchangeable with modules produced by competitors. Though companies are reluctant to make agreements regarding components with independent standardization agencies, once organizations consent, they are usually willing to extend the standards beyond the three compulsory areas listed above (Szyperski 1997). Similarly, the proposed architecture contains suggestions for features beyond the minimal requirements of component specification, interface, and distribution infrastructure. These additional features, though not essential, enhance a system's performance.

Selection of Features for Architecture. The next activity is to select a set of features for the proposed architecture. The set of features was derived by applying the two initial stages of review and synthesis of prior systems. Figure 6.1 divides these selected features into three categories: (1) general component structure, (2) interface, and (3) distribution infrastructure.

General Component Structure. The features selected for this category are the following: (1) standard components providing plug-and-play interoperability, (2) business domain process mapping directly to software, (3) encapsulation, (4) high cohesion, (5) the server depicting large business application components, and (6) a client as a glue component. These features describe the overall structure of large components.

A standardization of implementation is accomplished through s*tandard components providing plug-and-play interoperability*. This assures that a component performs as specified, in contrast to the standardization of interfaces, which guarantees that components interconnect properly. Identical specifications are required to have true interchangeability of elements, not solely identical interfaces. A developer needs to know what specific functionality to expect from a large component. Components with indistinguishable specifications must behave in the same way. The specification, however, implies nothing about the similitude of component implementations. Explicitly specified in the Stanford CHAIMS architecture, and implicit in several of the other systems (Helton 1999), is the assumption that the architecture will have incompatible components requiring wrappers to make them work together effectively. Commercial developers encounter a staggering quantity of incompatible software elements in their work, and these professionals are quite successful in devising mechanisms to enable these heterogeneous elements to function together to some degree. Inherently incompatible components, however, suggest sub-par performance, and will not be specified as a feature of the proposed architecture. Standard, compatible components are the ideal, and thus will be included.

The next feature selected for the architecture is *business domain process mapping directly to software*. This key concept has its roots in OO development facilitates preliminary data modeling, which is beyond the scope of this paper, and makes the software more comprehensible. This feature enables the software to capture the essence of what is actually happening in the corresponding business domain. One-to-one mapping of domain objects upon the software is well established in OO programming. Though mapping business elements to software is a key OO feature, process-oriented development (Matthes et al. 1999) extends this idea to elements of larger granularity. In development with large components, generally an entire business process,

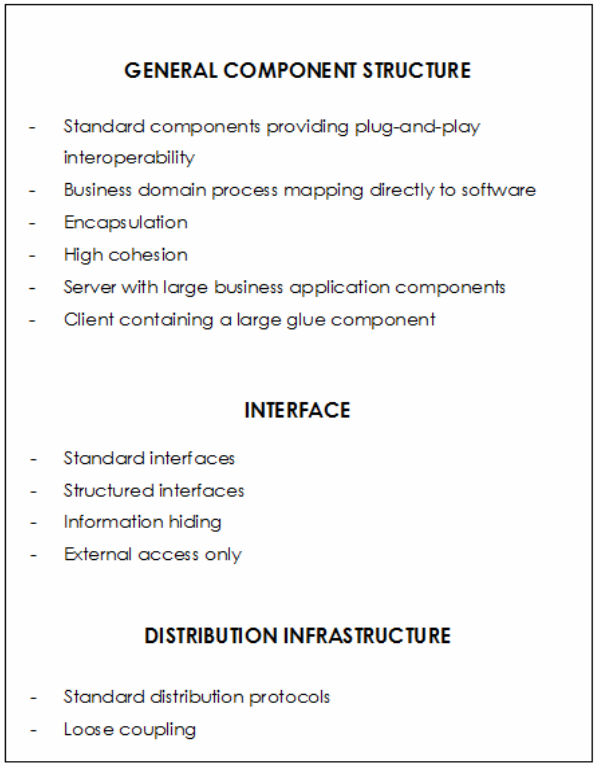


Figure 1 Selected Features for Large Component Systems

such as a hotel reservation system, is mapped to an enterprise component, consistent with process-oriented approach. Conceptually, this is appealing, because a

large system may be developed by combining just a handful of these coarse-grained elements.

The *encapsulation* feature, a notion associated with OO development helps make a component a manageable, autonomous unit. Unfortunately, this feature has remained to some degree an ideal, not having been implemented completely in practice. Instead of a module containing all of its related behavior, portions of its functionality would be placed outside of the module, or even in a separate library. At times, encapsulation is challenging, such as in the case of data storage. Conceptually, the handling of persistent data occurs within the module, whereas in reality the management of permanent data takes place mainly in an external DBMS. Physically, the data are stored in an external database shared by many enterprise components. Logically, however, the data are internal to the component in question. Though this and other considerations force designers to place, outside of a module, items properly belonging within it, encapsulation of everything directly pertaining to a module remains the ideal.

Proponents of structured programming established *high cohesion* of software modules as one of their guiding principles. This feature also fits well within OO design. Focusing everything within a component to one objective is readily attainable within small components with limited functionality. Applying this principle to large components is more challenging because of the increased complexity. Superficially, a highly cohesive large component may appear to contain elements with diverse functionality, which actually are related functions, all supporting a common purpose. Care must be taken to insure that items within a complicated component do not support unrelated tasks.

The next feature, a *server with large business application components*, has precedents in the Stanford University CHAIMS architecture and in the client/server application model. At least conceptually, ERP systems and other architectures also imply the presence of this feature, which implies a system with only a few components of coarse granularity. Tedious, yet relatively stable business processes, such as payroll and inventory, should make good candidates for mapping to the software as large server components. As indicated in Figure 1, the last feature from the general component structure category, *a client containing a large glue component*, was suggested by D'Souza and Wills (1998) and also incorporated into the Stanford CHAIMS architecture. An earlier precedent for this was the client/server application model, where a coordinating client requests services from a server module. Later, this feature will be discussed further, under the next step of the systems development research methodology, involving functionalities of components and their interactions.

Interface. The features chosen for this category, enumerated in Figure 1, are: (1) standard interfaces, (2) structured interfaces, (3) information hiding, and (4) external access only. Interfaces are crucial to the interrelationship of components.

Standard interfaces, the first feature in this category chosen for the proposed architecture, is required for large components to connect together correctly. Interchangeability between two components, as explained under the general component structure category, above, also demands functionality to be identical. Otherwise, a component would plug into the system properly, but not provide the desired functionality. Standard interfaces also offer extensibility to the system. A new component with a standard interface may be added to the system, as long as the new item's business functionality is compatible with the rest of the application.

The proposed architecture for large components would incorporate *structure interfaces* to eliminate the possibility of generating confusing interface inheritance hierarchies through using OO interfaces, such as those featured in several systems described earlier (Enterprise JavaBeans, Harvard health information systems, SAP R/3). The Stanford CHAIMS project has demonstrated that large, stable components work well with straightforward structured interfaces rather adding the complications of OO interfaces. Large, stable components do not require the dynamic capability of instantiating interfaces from classes of interfaces.

The next feature incorporated into the proposed software architecture is *information hiding*, which abstracts implementation details behind a component's interface. This concept was common in structured programming and was carried over to some extent in OO development. The object approach, unfortunately, permitted direct external access to nested interfaces, thereby completely nullifying the hiding feature.

Permitting *external access only* to components is a feature related to information hiding that will be included in the proposed architecture. As suggested by Lewandowski (1998), access to embedded interfaces should be only through delegation. This forwarding of calls maintains information hiding, which is important for keeping large components manageable.

Distribution Infrastructure. As indicated in Figure 1, the features in this category are: (1) standard distribution protocols, and (2) loose coupling. These features enable remote communication between components, an expectation in current large business applications.

Standard distribution protocols are obligatory for any intercommunication whatsoever between physically isolated modules. The three most widely used commercial systems, CORBA, DCOM, and Java, were designed for small components. Enterprise JavaBeans, is an extension of Java that handles large components. EJB, however, overextends the OO paradigm, retaining features that worked satisfactorily with FGCs, but which

are unsuited for components of large granularity. CBSD demands attention on the business application, not the underlying distributions structure. Consequently, the developer usually is required to use the most suitable distribution system available, irrespective of problems it might present with large components. The Stanford CHAIMS research project managed to achieve communication between large components by creating a new middleware layer to set atop a current commercial distribution system. This is a sub-optimal solution for communication among large communications. The architecture, therefore, includes a suggestion for a distribution system specifically designed for large components.

Loose coupling, a key notion borrowed from structured programming (Sametinger 1997), is the final feature recommended for the improved architecture for assembling components of large granularity. If not loosely coupled, independent deployment of components is infeasible. Even moderate coupling complicates the assembly of components as autonomous units. Instead of building software from discrete component blocks, the developer must wire together entire sub-assemblies of overly coupled modules.

3 Definition of Component Functionalities and Interrelationships

During this step in creating the system architecture, functionalities of system components and interrelationships among them will be defined. In the previous step, review and synthesis of prior systems, desirable features were derived for an improved architecture for composing large components, as indicated in Figure 1. This step will focus upon what components do and how they relate to other system elements.

Component Functionality. As discussed in the previous phase, the proposed architecture features standardized components of large granularity. This means that the components from one company are completely interchangeable with those from another, as long as the elements have identical specifications and interfaces. The system may be extended by adding other compatible components. Thus the architecture calls for homogeneous elements because research with the systems discussed in Helton (1999) indicates serious problems with handling disparate elements. These standardized elements are included in the architecture as an ideal, though extenuating circumstances might force someone implementing such a system to use wrappers to harmonize the functioning of heterogeneous components. Entire business processes are mapped directly to the software as server components of coarse granularity. Thus, the entire application might be constructed from just a handful of large components, as discussed in the previous step on review and synthesis of prior systems.

A glue component coordinates the operations of the application, consistent with the Stanford CHAIMS project and suggested by D'Souza and Wills (1998). Since current technology assumes the use of graphical user interfaces, a user interface component may be added. Optionally, the glue component may be combined with the user interface component.

Irrespective of its type, each component encapsulates its functionality so that each component remains a discrete, manageable unit. A conceptual framework might abstract functionality in such a way that a data storage facility might seem to be encapsulated within the component. The software architecture, however, depicts the reality that the database management system is a distinct mechanism, external to the other components.

In addition, all components, regardless of their type, must exhibit high cohesion, as explained in the review and synthesis step of the research methodology. Everything within the enterprise component should focus upon one central objective.

Component Interrelationships. Interconnections between components involve two aspects of a system: (1) interfaces, and (2) the distribution infrastructure. These will be described in the following section and elaborated further in the third step, system architecture development, of the research methodology.

Standard interfaces are included in the improved architecture to make components plug compatible. Two components with identical specifications should be completely interchangeable if their interfaces are also indistinguishable. Standard interfaces may also be used to extend system functionality by adding components. The standard interfaces permit these appended components to fit properly into the system, contingent upon the component specification being compatible with the rest of the system.

The proposed architecture incorporates structured interfaces. In several of the systems studied (Helton (1999), OO interfaces were employed, rendering them susceptible to the generation of puzzling inheritance interface hierarchies.

The proposed architecture uses information hiding to abstract the implementations of large components, making them manageable. The inner workings may follow any paradigm whatsoever.

The only type of direct access to components permitted in the proposed architecture is direct external access. This restriction reinforces information hiding. Delegation permits indirect access to interior interfaces, as suggested by Lewandowski (1998).

Standard distribution protocols are specified in the proposed architecture, even though the only distribution systems commercially available either were designed for small components or have been extended somewhat imperfectly to handle large components. As was discussed in the review and synthesis of prior systems step of the research methodology, the distribution system ideally should be one that was specifically crafted to handle large components. A specialized infrastructure for large components would obviate the need for building extra middleware software, as the Stanford CHAIMS researchers were forced to do, to adapt large components to infrastructures really designed for fine-grained components.

Since the proposed architecture is for a distributed system, loose coupling of elements is required. As adherents of structured programming demonstrated, tight coupling of modules even complicates applications that are not distributed (Sametinger 1997). The glue component provides central control, which facilitates loose coupling. This will be discussed further in the next section.

4 System Architecture Specification

The final step in defining the improved software architecture is to bring everything together to form a coherent system. Though the focus here is upon composing large components for a business application, the architecture specifies some general requirements for the physical distribution of software. The system architecture provides detail down to the level of interface specification. The implementation of components is abstracted because component inner structure is a widely researched topic, beyond the scope of this dissertation. The overall system architecture is depicted in Figure 2. This figure conforms to the completed conceptual framework developed in (Helton 1999).

The proposed system architecture, as illustrated in Figure 2, contains a glue component, one or more large application server components, a database management system, and an underlying distribution infrastructure. The role of each of these elements in the architecture will be discussed in the subsequent material.

Glue Component. The glue component coordinates the activities of the large application components. The glue component, as well as the other components, has a structured interface. Observe in Figure 2 that all invocations are made through the glue component in forcing loose coupling between application elements. The effectiveness of this type of glue component has been well established in the client/server application model for programming (Berson 1996).

Server Application Components. Components A and B in Figure 2 represent a small number of large components that comprise the server application components. Their implementations are obscured through information hiding. No direct external access to inner elements is permitted. Either A or B, however, may have nested interfaces accessible through delegation, as described earlier in this chapter

A new server component X could be substituted for component B if the interfaces and specifications for both X and B are identical. This mechanism provides interchangeability of elements obtained from competing vendors. An element C, moreover, distinct in its functionality from anything presently available within the application, could be appended to the system to extend the application's capabilities. Component C would have to have a standard interface and its functionality could not conflict with that of the rest of the system.

As indicated in Figure 2, component A should not make a direct invocation to B, or vice versa, though this is possible theoretically. Having one server component call another violates the programming principle of passing unconditional control to another module. Channelling all calls through the glue component eliminates this type of undesirable invocation.

Database Management System. Though conceptually data is contained within components, at the operational level persistence requires components to access the database management system, as depicted in Figure 2.

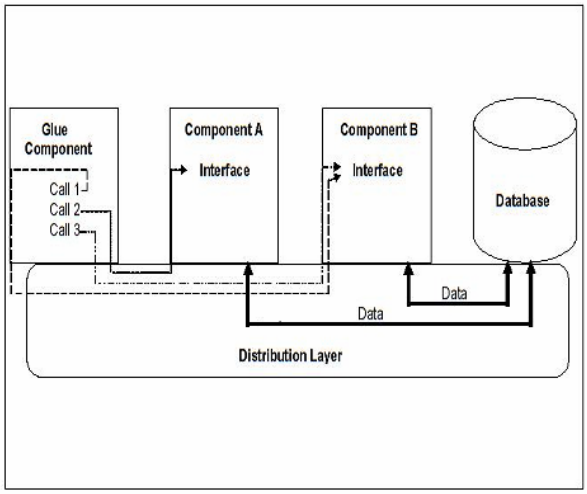


Figure 2 Coarse-Grained Component Architecture

The client component (Glue Component, in this case) makes invocations of server Components A and B as if they stored the data and, via delegation, the server components make calls to the DBMS.

Distribution System. In the proposed system architecture, a standard underlying distribution system manages all communication between the dispersed system elements. Ideally, an infrastructure specifically designed to handle large components would be used. This would eliminate the need for creating and maintaining, between the distribution system and the application, an additional software layer, such as the extra software stratum required in the Stanford CHAIMS system because the distribution system is designed to manage small rather than large components.

5 Summary

The goal of this study was to suggest an enhanced system architecture for large components. The conceptual framework (Helton 1999) formed the input for delineating the system architecture. This resulting software architecture synthesizes the best features from existing systems presented in (Helton 1999, Kim 2002) and related technologies.

References

[1] Bandini S., Paoli F., Manzoni S. & Mereghetti P. (2002) A supports system to COTS-based software development for business services. *14th International Conference On Software Engineering & Knowledge Engineering*, Ischia, Italy.

[2] Beringer D., Tornabene C., Jain P., & Wiederhold G. (1998) A language and system for composing autonomous, heterogeneous and distributed megamodules. *International Workshop on Large-Scale Software Composition*, in conjunction with DEXA'98, Ninth International Workshop on Database and Expert Systems Applications, Vienna, Austria, p. 826-833.

[3] Berson A. (1996) *Client/Server Architecture*, 2nd ed. New York: McGraw-Hill.

[4] Buck-Emden R. & Jürgen G. (1996) *SAP R/3 System: A Client/Server Technology*. Weinland A. (trans.). Harlow, England: Addison-Wesley.

[5] Budgen D. (2003) *Software Design*, 2nd ed. Harlow, England: Addison-Wesley.

[6] Carney D. (1997) Assembling large systems from COTS components: Opportunities, cautions, and complexities. SEI Monographs on Use of Commercial Software in Government Systems. Pittsburgh, PA: Carnegie Mellon University.

[7] Deibel S. & Greenes R. (1995) Component based computing in radiology systems architecture. Decision Systems Group Technical Report, DSG-AR-005-1.0, Brigham & Women's Hospital. Cambridge, MA: Harvard University.

[8] D'Souza D. & Wills A. C. (1998) *Objects, components and frameworks with UML: the Catalysis approach*. Reading, MA: Addison-Wesley.

[9] Helton D. (1999) Coarse-grained components as an alternative to component frameworks. *Proceedings, 4th International Workshop on Component-Oriented Programming*, Lisbon, Portugal.

[10] Hissam S. A., Seacord R. C. & Lewis G.A. (2002) Building systems from commercial components. *Proceedings of the 24th International Conference on Software Engineering*, Orlando, Florida, USA, p. 679-680.

[11] Inverardi P. & Tivoli M. (2002) The role of architecture in components assembly. *Proceedings, 7th International Workshop on Component-Oriented Programming*, Malaga, Spain.

[12] Kim S. D. (2002) Lessons learned from a nationwide CBD promotion project. *Communications of the ACM*, 45, no. 10, p. 83-87.

[13] Levi K. & Arsanjani A. (2002) A goal-driven approach to enterprise component identification and specification. *Communications of the ACM*, 45, no. 10, p. 45-52.

[14] Lewandowski S. M. (1998) Frameworks for component-based client/Server computing," *ACM Computing Surveys*, 30, no. 1, p. 3-27.

[15] Matthes, F., Wegner H. & Hupe P. (1999) A process-oriented approach to software component definition," *Proceedings, 1lth Conference on Advanced Systems Engineering*, Heidelberg, Germany, June 14-18, 1999.

[16] Pressman R. S. (1997) *Software Engineering: a Practitioner's approach*, 4th ed. New York: McGraw-Hill.

[17] Ravichandran T. & Rothenberger M. A. (2003) Software reuse strategies and component markets. *Communications of the ACM*, 46, no. 8, p. 109-114.

[18] Sametinger J. (1997) *Software Engineering with Reusable Components*. Berlin: Springer-Verlag.

[19] SAP AG (1997) R/3: System benefits of the business framework. Walldorf, Germany: SAP AG Technology Marketing.

[20] Shaw M. et al. (1995) Abstractions for software architecture and tools to support them. *IEEE Transactions on Software Engineering*, 21, no. 4, p. 314-335.

[21] Szyperski C. (1997) *Component Software: Beyond Object-Oriented Programming*. Reading, MA: Addison-Wesley.

[22] Thomas, A. (1997) Enterprise JavaBeans: server Component model for Java. Boston, MA: Patricia Seybold Group.

[23] Vitharana P. (2003) Risks and challenges of component-based software development. *Communications of the ACM*, 46, no. 8, p. 67-72.

JOŽEF STEFAN INSTITUTE

Jožef Stefan (1835-1893) was one of the most prominent physicists of the 19th century. Born to Slovene parents, he obtained his Ph.D. at Vienna University, where he was later Director of the Physics Institute, Vice-President of the Vienna Academy of Sciences and a member of several scientific institutions in Europe. Stefan explored many areas in hydrodynamics, optics, acoustics, electricity, magnetism and the kinetic theory of gases. Among other things, he originated the law that the total radiation from a black body is proportional to the 4th power of its absolute temperature, known as the Stefan–Boltzmann law.

The Jožef Stefan Institute (JSI) is the leading independent scientific research institution in Slovenia, covering a broad spectrum of fundamental and applied research in the fields of physics, chemistry and biochemistry, electronics and information science, nuclear science technology, energy research and environmental science.

The Jožef Stefan Institute (JSI) is a research organisation for pure and applied research in the natural sciences and technology. Both are closely interconnected in research departments composed of different task teams. Emphasis in basic research is given to the development and education of young scientists, while applied research and development serve for the transfer of advanced knowledge, contributing to the development of the national economy and society in general.

At present the Institute, with a total of about 700 staff, has 500 researchers, about 250 of whom are postgraduates, over 200 of whom have doctorates (Ph.D.), and around 150 of whom have permanent professorships or temporary teaching assignments at the Universities.

In view of its activities and status, the JSI plays the role of a national institute, complementing the role of the universities and bridging the gap between basic science and applications.

Research at the JSI includes the following major fields: physics; chemistry; electronics, informatics and computer sciences; biochemistry; ecology; reactor technology; applied mathematics. Most of the activities are more or less closely connected to information sciences, in particular computer sciences, artificial intelligence, language and speech technologies, computer-aided design, computer architectures, biocybernetics and robotics, computer automation and control, professional electronics, digital communications and networks, and applied mathematics.

The Institute is located in Ljubljana, the capital of the independent state of S**love**nia (or S♡nia). The capital today is considered a crossroad between East, West and Mediterranean Europe, offering excellent productive capabilities and solid business opportunities, with strong international connections. Ljubljana is connected to important centers such as Prague, Budapest, Vienna, Zagreb, Milan, Rome, Monaco, Nice, Bern and Munich, all within a radius of 600 km.

In the last year on the site of the Jožef Stefan Institute, the Technology park "Ljubljana" has been proposed as part of the national strategy for technological development to foster synergies between research and industry, to promote joint ventures between university bodies, research institutes and innovative industry, to act as an incubator for high-tech initiatives and to accelerate the development cycle of innovative products.

At the present time, part of the Institute is being reorganized into several high-tech units supported by and connected within the Technology park at the Jožef Stefan Institute, established as the beginning of a regional Technology park "Ljubljana". The project is being developed at a particularly historical moment, characterized by the process of state reorganisation, privatisation and private initiative. The national Technology Park will take the form of a shareholding company and will host an independent venture-capital institution.

The promoters and operational entities of the project are the Republic of Slovenia, Ministry of Science and Technology and the Jožef Stefan Institute. The framework of the operation also includes the University of Ljubljana, the National Institute of Chemistry, the Institute for Electronics and Vacuum Technology and the Institute for Materials and Construction Research among others. In addition, the project is supported by the Ministry of Economic Relations and Development, the National Chamber of Economy and the City of Ljubljana.

Jožef Stefan Institute
Jamova 39, 1000 Ljubljana, Slovenia
Tel.:+386 1 4773 900, Fax.:+386 1 219 385
Tlx.:31 296 JOSTIN SI
WWW: http://www.ijs.si
E-mail: matjaz.gams@ijs.si
Contact person for the Park: Iztok Lesjak, M.Sc.
Public relations: Natalija Polenec

INFORMATICA

AN INTERNATIONAL JOURNAL OF COMPUTING AND INFORMATICS

INVITATION, COOPERATION

Submissions and Refereeing

Please submit three copies of the manuscript with good copies of the figures and photographs to one of the editors from the Editorial Board or to the Contact Person. At least two referees outside the author's country will examine it, and they are invited to make as many remarks as possible directly on the manuscript, from typing errors to global philosophical disagreements. The chosen editor will send the author copies with remarks. If the paper is accepted, the editor will also send copies to the Contact Person. The Executive Board will inform the author that the paper has been accepted, in which case it will be published within one year of receipt of e-mails with the text in Informatica LaTeX format and figures in `.eps` format. The original figures can also be sent on separate sheets. Style and examples of papers can be obtained by e-mail from the Contact Person or from FTP or WWW (see the last page of Informatica).

Opinions, news, calls for conferences, calls for papers, etc. should be sent directly to the Contact Person.

QUESTIONNAIRE

☐ Send Informatica free of charge

☐ Yes, we subscribe

Please, complete the order form and send it to Dr. Drago Torkar, Informatica, Institut Jožef Stefan, Jamova 39, 1111 Ljubljana, Slovenia.

Since 1977, Informatica has been a major Slovenian scientific journal of computing and informatics, including telecommunications, automation and other related areas. In its 16th year (more than ten years ago) it became truly international, although it still remains connected to Central Europe. The basic aim of Informatica is to impose intellectual values (science, engineering) in a distributed organisation.

Informatica is a journal primarily covering the European computer science and informatics community - scientific and educational as well as technical, commercial and industrial. Its basic aim is to enhance communications between different European structures on the basis of equal rights and international refereeing. It publishes scientific papers accepted by at least two referees outside the author's country. In addition, it contains information about conferences, opinions, critical examinations of existing publications and news. Finally, major practical achievements and innovations in the computer and information industry are presented through commercial publications as well as through independent evaluations.

Editing and refereeing are distributed. Each editor can conduct the refereeing process by appointing two new referees or referees from the Board of Referees or Editorial Board. Referees should not be from the author's country. If new referees are appointed, their names will appear in the Refereeing Board.

Informatica is free of charge for major scientific, educational and governmental institutions. Others should subscribe (see the last page of Informatica).

ORDER FORM – INFORMATICA

Name: .....

Title and Profession (optional): .....

.....

Home Address and Telephone (optional): .....

.....

Office Address and Telephone (optional): .....

.....

E-mail Address (optional): .....

Signature and Date: .....

Informatica WWW:

http://ai.ijs.si/informatica/

Referees:

Witold Abramowicz, David Abramson, Adel Adi, Kenneth Aizawa, Suad Alagić, Mohamad Alam, Dia Ali, Alan Aliu, Richard Amoroso, John Anderson, Hans-Jurgen Appelrath, Iván Araujo, Vladimir Bajič, Michel Barbeau, Grzegorz Bartoszewicz, Catriel Beeri, Daniel Beech, Fevzi Belli, Simon Beloglavec, Sondes Bennasri, Francesco Bergadano, Istvan Berkeley, Azer Bestavros, Andraž Bežek, Balaji Bharadwaj, Ralph Bisland, Jacek Blazewicz, Laszlo Boeszoermenyi, Damjan Bojadžijev, Jeff Bone, Ivan Bratko, Pavel Brazdil, Bostjan Brumen, Jerzy Brzezinski, Marian Bubak, Davide Bugali, Troy Bull, Sabin Corneliu Buraga, Leslie Burkholder, Frada Burstein, Wojciech Buszkowski, Rajkumar Bvyya, Giacomo Cabri, Netiva Caftori, Particia Carando, Robert Cattral, Jason Ceddia, Ryszard Choras, Wojciech Cellary, Wojciech Chybowski, Andrzej Ciepielewski, Vic Ciesielski, Mel Ó Cinnéide, David Cliff, Maria Cobb, Jean-Pierre Corriveau, Travis Craig, Noel Craske, Matthew Crocker, Tadeusz Czachorski, Milan Češka, Honghua Dai, Bart de Decker, Deborah Dent, Andrej Dobnikar, Sait Dogru, Peter Dolog, Georg Dorfner, Ludoslaw Drelichowski, Matija Drobnič, Maciej Drozdowski, Marek Druzdzel, Marjan Družovec, Jozo Dujmović, Pavol Ďuriš, Amnon Eden, Johann Eder, Hesham El-Rewini, Darrell Ferguson, Warren Fergusson, David Flater, Pierre Flener, Wojciech Fliegner, Vladimir A. Fomichov, Terrence Forgarty, Hans Fraaije, Stan Franklin, Violetta Galant, Hugo de Garis, Eugeniusz Gatnar, Grant Gayed, James Geller, Michael Georgiopolus, Michael Gertz, Jan Goliński, Janusz Gorski, Georg Gottlob, David Green, Herbert Groiss, Jozsef Gyorkos, Marten Haglind, Abdelwahab Hamou-Lhadj, Inman Harvey, Jaak Henno, Marjan Hericko, Henry Hexmoor, Elke Hochmueller, Jack Hodges, Doug Howe, Rod Howell, Tomáš Hruška, Don Huch, Simone Fischer-Huebner, Zbigniew Huzar, Alexey Ippa, Hannu Jaakkola, Sushil Jajodia, Ryszard Jakubowski, Piotr Jedrzejowicz, A. Milton Jenkins, Eric Johnson, Polina Jordanova, Djani Juričič, Marko Juvancic, Sabhash Kak, Li-Shan Kang, Ivan Kapustøk, Orlando Karam, Roland Kaschek, Jacek Kierzenka, Jan Kniat, Stavros Kokkotos, Fabio Kon, Kevin Korb, Gilad Koren, Andrej Krajnc, Henryk Krawczyk, Ben Kroese, Zbyszko Krolikowski, Benjamin Kuipers, Matjaž Kukar, Aarre Laakso, Sofiane Labidi, Les Labuschagne, Ivan Lah, Phil Laplante, Bud Lawson, Herbert Leitold, Ulrike Leopold-Wildburger, Timothy C. Lethbridge, Joseph Y-T. Leung, Barry Levine, Xuefeng Li, Alexander Linkevich, Raymond Lister, Doug Locke, Peter Lockeman, Vincenzo Loia, Matija Lokar, Jason Lowder, Kim Teng Lua, Ann Macintosh, Bernardo Magnini, Andrzej Małachowski, Peter Marcer, Andrzej Marciniak, Witold Marciszewski, Vladimir Marik, Jacek Martinek, Tomasz Maruszewski, Florian Matthes, Daniel Memmi, Timothy Menzies, Dieter Merkl, Zbigniew Michalewicz, Armin R. Mikler, Gautam Mitra, Roland Mittermeir, Madhav Moganti, Reinhard Moller, Tadeusz Morzy, Daniel Mossé, John Mueller, Jari Multisilta, Hari Narayanan, Jerzy Nawrocki, Rance Necaise, Elzbieta Niedzielska, Marian Niedq'zwiedziński, Jaroslav Nieplocha, Oscar Nierstrasz, Roumen Nikolov, Mark Nissen, Jerzy Nogieć, Stefano Nolfi, Franc Novak, Antoni Nowakowski, Adam Nowicki, Tadeusz Nowicki, Daniel Olejar, Hubert Österle, Wojciech Olejniczak, Jerzy Olszewski, Cherry Owen, Mieczyslaw Owoc, Tadeusz Pankowski, Jens Penberg, William C. Perkins, Warren Persons, Mitja Peruš, Fred Petry, Stephen Pike, Niki Pissinou, Aleksander Pivk, Ullin Place, Peter Planinšec, Gabika Polčicová, Gustav Pomberger, James Pomykalski, Tomas E. Potok, Dimithu Prasanna, Gary Preckshot, Dejan Rakovič, Cveta Razdevšek Pučko, Ke Qiu, Michael Quinn, Gerald Quirchmayer, Vojislav D. Radonjic, Luc de Raedt, Ewaryst Rafajlowicz, Sita Ramakrishnan, Kai Rannenberg, Wolf Rauch, Peter Rechenberg, Felix Redmill, James Edward Ries, David Robertson, Marko Robnik, Colette Rolland, Wilhelm Rossak, Ingrid Russel, A.S.M. Sajeev, Kimmo Salmenjoki, Pierangela Samarati, Bo Sanden, P. G. Sarang, Vivek Sarin, Iztok Savnik, Ichiro Satoh, Walter Schempp, Wolfgang Schreiner, Guenter Schmidt, Heinz Schmidt, Dennis Sewer, Zhongzhi Shi, Mária Smolárová, Carine Souveyet, William Spears, Hartmut Stadtler, Stanislaw Stanek, Olivero Stock, Janusz Stokłosa, Przemysław Stpiczyński, Andrej Stritar, Maciej Stroinski, Leon Strous, Ron Sun, Tomasz Szmuc, Zdzislaw Szyjewski, Jure Šilc, Metod Škarja, Jiři Šlechta, Chew Lim Tan, Zahir Tari, Jurij Tasič, Gheorge Tecuci, Piotr Teczynski, Stephanie Teufel, Ken Tindell, A Min Tjoa, Drago Torkar, Vladimir Tosic, Wieslaw Traczyk, Denis Trček, Roman Trobec, Marek Tudruj, Andrej Ule, Amjad Umar, Andrzej Urbanski, Marko Uršič, Tadeusz Usowicz, Romana Vajde Horvat, Elisabeth Valentine, Kanonkluk Vanapipat, Alexander P. Vazhenin, Jan Verschuren, Zygmunt Vetulani, Olivier de Vel, Didier Vojtisek, Valentino Vranić, Jozef Vyskoc, Eugene Wallingford, Matthew Warren, John Weckert, Michael Weiss, Tatjana Welzer, Lee White, Gerhard Widmer, Stefan Wrobel, Stanislaw Wrycza, Tatyana Yakhno, Janusz Zalewski, Damir Zazula, Yanchun Zhang, Ales Zivkovic, Zonling Zhou, Robert Zorc, Anton P. Železnikar

Informatica

An International Journal of Computing and Informatics

Archive of abstracts may be accessed at USA: http://, Europe: http://ai.ijs.si/informatica, Asia: http://www.comp.nus.edu.sg/ liuh/Informatica/index.html.

Subscription Information Informatica (ISSN 0350-5596) is published four times a year in Spring, Summer, Autumn, and Winter (4 issues per year) by the Slovene Society Informatika, Vožarski pot 12, 1000 Ljubljana, Slovenia.
The subscription rate for 2004 (Volume 28) is
– USD 80 for institutions,
– USD 40 for individuals, and
– USD 20 for students
Claims for missing issues will be honored free of charge within six months after the publication date of the issue.

LaTeX Tech. Support: Borut Žnidar, Kranj, Slovenia.
Lectorship: Fergus F. Smith, AMIDAS d.o.o., Cankarjevo nabrežje 11, Ljubljana, Slovenia.
Printed by Biro M, d.o.o., Žibertova 1, 1000 Ljubljana, Slovenia.

Orders for subscription may be placed by telephone or fax using any major credit card. Please call Mr. R. Murn, Jožef Stefan Institute: Tel (+386) 1 4773 900, Fax (+386) 1 219 385, or send checks or VISA card number or use the bank account number 900–27620–5159/4 Nova Ljubljanska Banka d.d. Slovenia (LB 50101-678-51841 for domestic subscribers only).

Informatica is published in cooperation with the following societies (and contact persons):
Robotics Society of Slovenia (Jadran Lenarčič)
Slovene Society for Pattern Recognition (Franjo Pernuš)
Slovenian Artificial Intelligence Society; Cognitive Science Society (Matjaž Gams)
Slovenian Society of Mathematicians, Physicists and Astronomers (Bojan Mohar)
Automatic Control Society of Slovenia (Borut Zupančič)
Slovenian Association of Technical and Natural Sciences / Engineering Academy of Slovenia (Igor Grabec)
ACM Slovenia (Dunja Mladenič)

Informatica is surveyed by: AI and Robotic Abstracts, AI References, ACM Computing Surveys, ACM Digital Library, Applied Science & Techn. Index, COMPENDEX*PLUS, Computer ASAP, Computer Literature Index, Cur. Cont. & Comp. & Math. Sear., Current Mathematical Publications, Cybernetica Newsletter, DBLP Computer Science Bibliography, Engineering Index, INSPEC, Linguistics and Language Behaviour Abstracts, Mathematical Reviews, MathSci, Sociological Abstracts, Uncover, Zentralblatt für Mathematik

The issuing of the Informatica journal is financially supported by the Ministry of Education, Science and Sport, Trg OF 13, 1000 Ljubljana, Slovenia.

Volume 28 Number 3 November 2004 **ISSN 0350-5596**

Informatica

An International Journal of Computing and Informatics

Introduction	B. Vilfan, R. Grossi	**225**
An Average Running Time Analysis of a Backtracking Algorithm to Calculate the Size of the Union of Cartesian Products	S. Suzuki, T. Ibaraki	**227**
A Spectral Approach to Graphical Representation of Data	D. Bokal, M. Juvan, B. Mohar	**233**
Algorithms for Drawing Polyhedra from 3-Connected Planar Graphs	A. Orbanić, M. Boben, G. Jaklič, T. Pisanski	**239**
Grammar-Based Systems: Definition and Examples	M. Mernik, M. Črepinšek, T. Kosar, D. Rebernak, V. Žumer	**245**
Improved Error Recovery in Generated LR Parsers	B. Slivnik, B. Vilfan	**257**

Informational Design of Conscious Entities	A.P. Železnikar	**265**
The Demarcate Construction: A New Form of Tree-based Priority Queues	R.S.M. Goh, W.T. Tang, I.L.-J. Thng, M.T.R. Quieta	**277**
Fault-Free Maximal Submeshes in Faulty Torus-Connected Multicomputers	S.-M. Yoo, H.Y. Youn, H. Choo	**289**
Distributing State Space for Parallel Computation of CTL Model Checking	M. Bourahla, M. Benmohamed	**297**
On-line Handwriting Chinese Character Analysis and Recognition Using Stroke Correspondence Search	J. Shin	**307**
A System for Evaluation of Human Upper Extremity	N. Klopčar, J. Lenarčič	**315**
A Software Architecture for Enterprise Components	D.A. Helton	**323**

Informatica **28** (2004) Number 3, pp. 225–333