Anul L 2005

S T U D I A
UNIVERSITATIS BABEŞ-BOLYAI

INFORMATICA

1

Redacţia: 3400 Cluj-Napoca, str. M. Kogălniceanu nr. 1 Telefon 405300

SUMAR – CONTENTS – SOMMAIRE

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume L, Number 1, 2005

PROFESSOR MILITON FRENŢIU AT HIS SIXTIES

LEON ŢÂMBULEA

Professor Frenţiu has graduated Mathematics 38 years ago (the computation is simple, and requires no programming effort: 60 years of age minus 22 at faculty graduation). For seven years after graduation he specialised himself in Probability Theory and Numerical Analysis and started doctoral studies in Numerical Analysis. This background is remarked, as well, from his scientific activity, his first three published scientific papers, very well received in the literature, being from these domains. During this period I was a student attending the Probability Theory seminars of Professor Frenţiu, an assistant at that time.

The scientific and didactic activity of Professor Frenţiu have been radically changed in 1974, 31 years ago, when he was a beneficiary of a doctoral fellowship in England. Up to this moment, our faculty had very few study subjects in Computer Science, and the lectures could be taught only by our colleagues, Cornel Tarţia and Grigor Moldovan. The huge evolution of Computer Science during these thirty years is noticeable, as well, by comparing the number of academic positions at the Department of Computer Science, from the very few hours of Computer Science then to the 90 didactical positions of Computer Science today.

At his return from England, in 1977, with his awarded PhD Degree, the didactical activity has been oriented towards Computer Science. He was promoted as lecturer professor in 1979, in 1990 he became associate professor and for 10 years he has been a professor in Computer Science. He has to teach classes at many study objects, but the most important are related to algorithmics, programming, programs correctness, modeling and simulation, or the master lectures. We remember here his 26 published books and manuals, or the more than 50 scientific papers having him as author or co-author.

During the 40 years he is working with our Faculty, he realised many didactical and extradidactical tasks. I recall here admissions commisions, license commisions, doctoral coordinations, member of the faculty time-table team, the main promoter of the Computer Science issues of the Studia Universitatis journal, main promoter of the preprint journal of the Department of Computer Science, head of chair and head of department.

Professor Frenţiu added much energy and passion in all his achievements, he worked very hard for his department, his chair and his colleagues. I am asking myself whether this strong passion for his profession has any contribution to the fact that both children of the Frenţiu family are Computer Science graduates.

Received by the editors: April 15, 2005.

Since very long we have found Professor Frenţiu at his office all day long, during working days and week-ends as well, both in the ages when his office has been on the old August 23-rd St., at the Red Building, or at the new location in the University Campus. He had many lectures to prepare, many academic curricula to conceive and argue for, many student papers or projects to grade. The result may be seen, as well, by the large number of books and papers he has written, the large number of academic positions in our current department, but also, by the large number of colleagues that wished to participate to Professor Frenţiu 60 Years festivities. Colleagues that, I am certain, do appreciate all the energy and effort Professor Frenţiu has spent for the success of the Faculty of Mathematics and Computer Science in Cluj.

Published papers

(1) Frentiu M., Program Correctness in Software Engineering Education, Proceedings of the International Conference on Computers and Communications ICCCM4, pp.154-157, Oradea, 2004, may 27-29.
(2) Frentiu M., An Overview on Program Inspection, Proceedings of the Symposium "Zilele Academice Clujene", 2004, pp. 9-14
(3) Frentiu M., Formal Methods in Software Engineering Education, Proceedings ICELM1, Tg.Mures, 3-5 iunie 2004.
(4) Niculescu V., M. Frentiu, Designing Correct Parallel Programs from Specifications, "The 8th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2004)", Orlando, USA, July 18-21, 2004.
(5) Frentiu M., Ioan Lazar, and Horia F. Pop, On Individual Projects in Software Engineering Education, Studia Univ. Babes-Bolyai, Informatica, Volume XLVIII, Number 2, 2003, pp.83-94.
(6) Frentiu M., Formal Methods in Education, Colocviul Academic Clujean de INFORMATICA, Cluj-Napoca, iunie 2003, pp.7-12
(7) Lazarovici Gh., M.Lazarovici, M.Frentiu,Statistical Analysis of the archaeological material (Cucuteni A3 phase) from Ruginoasa settlement, Iasi district, Annals of the Tiberiu Popoviciu Seminar of Functional Equations, Approximations and Convexity, ISSN 1584-4536, vol.1, 2003, pp.169-186.
(8) Frentiu M., H.F.Pop, A Study of Dependence of Software Attributes Using Data Analysis Technique, Studia Univ. Babes-Bolyai, Seria Informatica, XLV(2002), 2, 53-66.
(9) Frentiu M., The Impact of Style on Program Comprehensibility, Proceedings of the Symposium "Zilele Academice Clujene", 2002, pp. 7-12
(10) Frentiu M., and H.F.Pop, A Study of Licence Examination Results Using Fuzzy Clustering Techniques, Research Seminar on Computer Science, 2001, pp. 99-106.
(11) Frentiu M., Is teaching formal methods really needed?, Research Seminar on Computer Science, 2000, pp. 57-64.

(12) Frentiu M., Teaching Program Correctness, Proceedings II AET Workshop, Cluj-Napoca, 4-10 October 2000, pp.232-246.
(13) Frentiu M., On programming style - program correctness relation, Studia Univ. Babes-Bolyai, Seria Informatica, XLV(2000), 2, 60-65.
(14) Frentiu M., Teaching Program Correctness, Proceedings II AET Workshop, Cluj-Napoca, 4-10 October 2000, pp.232-246.
(15) Frentiu M., On Program Correctness and teaching programming, Computer Science Journal of Moldova, vol.5(1997), no.3, 250-260.
(16) Frentiu M., D.Dumitrescu, B.Pârv, L.Tambulea si H.F.Pop, Algoritmi utili existenti în reteaua Universitatii, Research Seminar on Computer Science, 1999, pp. 111-122
(17) Frentiu M., Aspecte privind predarea programarii n liceu, Seminarul de Didactica Matematicii, Volumul 11(1995), 53-60.
(18) Frentiu M., An overview on Computer Science, Babes-Bolyai University, Faculty of Mathematics, Res. Sem., 1997, no.2, pp.1-2.
(19) Frentiu M., Is there a software crisis?, Babes-Bolyai University, Faculty of Mathematics, Res. Sem., 1997, no.2, pp.4-8.
(20) Frentiu M., Munteanu E., Nevertheless, there is a Computer Science, Studia Univ. Babes-Bolyai, Seria Informatica, 1(1996), 1, 1-6
(21) M.Frentiu, The impact of program correctness theory on teaching programming, în "First Joint Workshop on Modern Applied Mathematics", Ilieni, 12-16 June 1995
(22) Boian F., M.Frentiu, Z.Kasa, Folosirea calculatorului in predarea geometriei, in Lucrarile Conferintei "Informatizarea invatamantului", Balti, 4-7 octombrie 1995, pp.66-71.
(23) Frentiu M., Asupra predarii informaticii în liceu, in Lucrarile Conferintei "Informatizarea invatamantului", Balti, 4-7 octombrie 1995, pp.47-51.
(24) Frentiu M., Reguli de programare pentru incepatori, in Lucrarile Conferintei "Informatizarea invatamantului", Balti, 4-7 octombrie 1995, pp.55-61.
(25) Frentiu M., B.Parv, Programming proverbs revisited, Studia Univ. Babes-Bolyai, Math., XVIII, no.3, 1993, pp.49-58.
(26) Frentiu M., Consideratii privind predarea informaticii în liceu, Seminarul de Didactica Matematicii, Volumul 9, 1993, pp.93-96.
(27) Frentiu M., E.Munteanu, Past and Future in Computer Science at the University of Cluj, Babes-Bolyai University, Faculty of Mathematics, Res. Sem., 15(1993), no.5, pp.5-28, [MR94-1438806-01A73]
(28) Frentiu M., Gh. Lazarovici, Metode de clasificare automat în arheologie, Acta Musei Napocensis, XXIV-XXV, pp.909-918, Cluj-Napoca, 1992.
(29) Boian F., Frentiu M., Program testing for Loop-Exit Schemes, Studia Univ. "Babes Bolyai", Math., XXXVIII, 3, 1992, pp. 15-18 .
(30) Frentiu M., B. Prv, V. Prejmerean, Abstract Data Types for Increasing the Productivity in Programming, Babes-Bolyai Univ., Faculty of Mathematics, Res. Sem., 14(1992), no.5, pp. 6-11.

(31) Boian F., M.Frentiu, Z. Kasa, Parallel Executable Sequences in Serial Programs, în Studia Univ. "Babes Bolyai", Mathematica, XXXIV, 3, 1989, pp. 3-16.
(32) Frentiu M., Gh. Lazarovici, Seriation and relative chronology of archaeological complexes from Gornea, in "Second Romanian Conference on the application of Physics methods in archaeology", Cluj Napoca, February 17 18, 1989, Vol.2, pp.65-85.
(33) Frentiu M, J.Imreh, A.Motiu, Etude petrographique et geochimique des calcaires eocenes de la region situee au sud de Razoare, Studia Univ. Babes Bolyai, Geol., XXXIV, 1989, pp.59-67.
(34) Boian F., M.Frentiu, and Z.Kasa, Efficiency in parallel evaluation of Arithmetic Expressions, în "Babes Bolyai" University, Faculty of Mathematics, Seminar on Complexity, Preprint no. 10/1989
(35) Boian F., M.Frentiu, and Z.Kasa, Computer aided geometry, în "Babes Bolyai" University, Faculty of Mathematics, Seminar on Computer Science, Preprint no. 9/1989, pp. 11-20.
(36) Frentiu M, F.Boian, Z.Kasa, Parallel Execution in Loop Exit Schemes, în "Babes Bolyai" University, Faculty of Mathematics, Seminar on Computer Science, Preprint no. 9/1988, pp. 3-16.
(37) Frentiu M., and Gh.Lazarovici, Methods for automated classification used in archaeology. An application to neolithic graves and ornaments, in First Romanian Conference on the Applied Physics Methods in Archaeology, Vol.1(1988), pp.131-146.
(38) Frentiu M, F. Boian, Z. Kasa, L. Tambulea, Fortran must be improved, Studia Univ. Babes-Bolyai, Math., XXXII, 3, 1987, pp. 15-16.
(39) Imreh J., A.Motiu, Frentiu M, Etude geochimique de la dolomits des calcaires appartenent de la partie No du bassin de Transylvanie, în Lucrarile simpozionului National "The Eocene from the Transylvanian Basin", Roumania, Univ. Cluj Napoca, 1987, pp.207-219.
(40) Frentiu M, F.Boian, Z.Kasa, Folosirea calculatorului personal în predarea geometriei, în Lucrarile Seminarului "Didactica Matematică", vol.4 (1987, 88), pp.39-50.
(41) Frentiu M, F.Boian, Z.Kasa, L.Tambulea, Sistem de programe pentru elaborarea statelor de functii, în Lucrările sesiunii stiintifice a CCUB, Bucuresti, 1987, pp. 438-443.
(42) Frentiu M, F.Boian, Z.Kasa, Elemente de programare în limbajul BASIC. în Lucrările Seminarului "Didactica matematică", 1987 1988, pp. 51-64.
(43) Frentiu M, On the asymptotic aspect of the approximation of functions by means of the D.D.Stancu operators, Seminar on Numerical Analysis, preprint no.5, 1987, pp.57-64.
(44) Frentiu M, F. Boian, Folosirea corect a matricelor în programarea modulară, Lucrarile Seminarului "Didactica Matematic", Univ. Cluj Napoca, 1987, pp. 86-101.

(45) Frentiu M, F. Boian, Z. Kasa, A System for Program Writing and Debuging. În "Babes Bolyai" University, Faculty of Mathematics, Seminar on Computer Science, Preprint no. 5/1987, pp. 1-21.
(46) Imreh J., Frentiu M. si N.Meszaros, A Geochemical Study on the limestone at Răstoci, Studia Univ. Babes Bolyai, ser. Geologica, XXXI, 1, 1986, 15-26.
(47) Frentiu M, J.Imreh, si N.Meszaros, Geochemische untersuchungen uber eine eozan-oligozan kalkstein - serie aus dem norden des Siebenburgischen Beckens (Rumanien), Annales Univ. Sci.Budapestinensis, XXVI, 1986, pp. 13-30.
(48) Frentiu M., T.Chiorean, Determinarea prin simulare a capacităţii portante a elementelor de rezistent ale unei constructii, în Lucrrile celui de al V lea Colocviu de Informatica, Iasi, 18 19 octombrie 1985, pp. 786-788.
(49) Frentiu M., Program pentru prelucrări statistice în geologie, în Lucrările Laboratorului de Cercetari interdisciplinare ale Univ. Cluj Napoca, 19 decembrie 1985, pp. 29-34.
(50) F.Boian, Frentiu M., Z.Kasa si L.Tambulea, Towards a new standard Fortran, Seminar on Computer Science, preprint no.6/1985, pp.1-20.
(51) F.Boian, Frentiu M., Z. Kasa, L. Tâmbulea, l. Erdo, A. Szen, Simularea automatelor programabile, în Lucrările simpozionului "Informatica si aplicatiile sale", Zilele academice Clujene, Cluj Napoca, 1985, pp. 44-51.
(52) Frentiu M., On the program correctness, Seminar on Computer Science, preprint no.4, 1984, pp.75-84.
(53) Frentiu M., J.Szilagyi, The MACRO2 Macroprocessor, Mathematica, 22(1980), nr.2, 357-358.
(54) Frentiu M., C.Terchilă, Macroprocesor în sistemul SIRIS 2, Studia Univ. Babes Bolyai, 1979, nr.1, 71-72.
(55) Frentiu M., Macroprocesoare în scrierea translatoarelor, în Lucrările celui de al V lea Simpozion "Informatica si conducere", 5 12 mai 1979, pp.77-79.
(56) Frentiu M., Error correction in a simple precedence language, Mathematica, 20(43), nr.2, 1978 pp.159-162. [MR80a:68100-68F25].
(57) Frentiu M., A global error correcting parser, Mathematica, 19(42), nr.1, 1977, pp.41-43. [MR80a:68100-68F25]
(58) Coman Gh., Frentiu M., Bivariate spline approximation, Studia Univ. Babes Bolyai, 1974, nr.1, 59-64.
(59) Frentiu M., A method for generation of pseudo random numbers, Studia Univ. Babes Bolyai, 1974, nr.1, 41-43.
(60) Frentiu M., Combinatii liniare de polinoame Bernstein si de operatori Mirakyan, Studia Univ. Babes Bolyai, 1970, nr.1, 63-68.

Other papers

(1) Frentiu M., Error correcting codes, MPhil. thesis, Brunel Univ., 1975.
(2) Frentiu M., Some aspects of error correction of programming languages Ph.D. Thesis, Brunel Univ., London, 1977.
(3) M. Frentiu, Conceptul de subalgoritm în învatarea programarii, Gazeta de informatica, 2000.
(4) Frentiu M., F.Boian, Folosirea corecta a matricelor in programarea modulara, Lucrarile Seminarului "Didactica Matematica", Univ. Cluj- Napoca, 1987, pg.86-101
(5) M.Frentiu, Asupra rezolvarii problemelor date la admiterea la Facultatea de Matematica si Informatica a Univ. Babes-Bolyai, 22 iulie 2002, Gazeta de informatica, 2002.

Published books

(1) Gh.Coman, M.Frentiu, Introducere in Informatică (Introduction to computers and programming), Ed. Dacia, 1982, 213 pagini.
(2) M.Frentiu, B.Prv, Elaborarea programelor. Metode si tehnici moderne, Ed. Promedia, Cluj-Napoca, 1994, 208 pagini.
(3) M.Frentiu si altii, Manualul începatorului în Programarea Pascal, Ed. Microinformatica, Cluj-Napoca, 1995, 252 pagini, I.S.B.N.973-9215-04-1.
(4) M.Frentiu si altii, Programare Pascal. Programe ilustrative, probleme propuse, pentru elevi si studenti, Ed. Promedia, 1995, 229 pagini, I.S.B.N. 973-96862-1-4.
(5) M.Frentiu, I.Lazar, S.Motogna si V.Prejmereanu, Elaborarea algoritmilor, Ed.Universitatii Babes-Bolyai, Cluj-Napoca, 1998, 188 pagini, ISBN 973-9261-16-7.
(6) M.Frentiu, I.Lazar, S.Motogna si V.Prejmereanu, Programare Pascal, Ed.Universitatii Babes-Bolyai, Cluj-Napoca, 1998, 392 pagini, ISBN 973-9261-18-3.
(7) I. Lazăr, M.Frentiu, V.Niculescu, Programare orientata obiect in Java, Ed. Univ. Petru Maior, Tg.Mures, 1999, 283 pagini , ISBN 973-99054-8-X
(8) M.Frentiu, I.Lazar, Bazele Programarii: Proiectarea Algoritmilor, 2000, Ed. Univ. Petru Maior, Tg.Mures 184 pagini ISBN 973-8084-06-7
(9) M.Frentiu, Verificarea Corectitudinii Programelor, Ed.Univ."Petru-Maior", Tg.-Mures, 2001, 116 pagini, ISBN 973-8084-32-6
(10) M.Frentiu, I.Lazar, L.Tambulea, F.Boian, Informatica de baza, Ed.Universitatii Babes-Bolyai, Cluj-Napoca, 2005, 226 pagini, ISBN 973-610-340-4

Manuals and other publications of the same nature

(1) M.Frentiu et al, Informatica pentru elevi, Ed.Microinformatica, Editiile 1-2 în 1992 si Editia 3 în 1993, 210 pagini.

(2) M.Frentiu, I.Lazar, S.Motogna si V.Prejmereanu, Elaborarea algoritmilor, Ed.Universitatii Babes-Bolyai, Cluj-Napoca, 1996, 188 pagini.
(3) M.Frentiu, I.Lazar, S.Motogna si V.Prejmereanu, Programare Pascal, Ed.Universitatii Babes-Bolyai, Cluj-Napoca, 1996, 392 pagini.
(4) Fl. Boian, M.Frentiu, S. Groze, E.Iacob, S.Iurian, E.Iacob, Kasa Z., S.Motogna, H.F.Pop, V.Prejmereanu, L.Tâmbulea, Bazele Informaticii I. Culegere de probleme pentru lucrarile de laborator (editia a II-a), 1992, 182 pagini.
(5) M. Frentiu, Z. Kasa, L.Tambulea, C. Tartia, Utilizarea calculatorului personal în liceu, 1988, 120 pagini.
(6) M.Frentiu, Z.Kasa, L.Tambulea, C. Tartia Utilizarea calculatorului personal PRAE M, 1986, 81 pagini.
(7) M.Frentiu, Bazele Matematice ale Calculatoarelor, Univ. "Avram Iancu", Cluj-Napoca, 1993.
(8) Fl. Boian, M.Frentiu, S. Groze, D.Dumitrescu, L.Lupsa, Kasa Z., L.Tâmbulea, Bazele Informaticii I. Culegere de probleme pentru lucrarile de laborator, Ed.I-a, 1982, 147 pagini.
(9) M.Frentiu, F.Boian, Bazele Informaticii. Limbajul Pascal, (editia a II-a, 1992), 190 pagini.
(10) M.Frentiu, S.Groze, Informatica, Litografia Univ. D. Cantemir, Cluj-Napoca, 1992, 136 pagini.
(11) M.Frentiu, F.Boian, Bazele Informaticii. Limbajul Pascal, (editia a I-a, 1990), 185 pagini.
(12) M.Frentiu, Geologie matematica, 1987, 122 pagini.
(13) M.Frentiu, S.Groze, Bazele Informaticii I, 1983, 202 pagini.
(14) M.Frentiu, S.Groze, Bazele Informaticii (editia a II-a), 1986, 243 pagini.
(15) M.Frentiu, S.Groze, Programare si Informatica, 1982, 193 pagini.
(16) M.Frentiu, F.Boian, Bazele informaticii I. Instruire în programare, 1980, 117 pagini.
(17) M.Frentiu, Sisteme de operare si teleprelucrarea datelor. Assiris. 1979, 80 pp.
(18) M.Frentiu si colectiv, Bibliotecile MATH I si MATH II pentru calculatoarele personale, cu ITCI, 1987 1988.

Participations at research programmes financed by national institutions

(1) Studiul elaborarii unei retele a Universitatii. Studiul complexitatii algoritmilor si optimizarea programelor (beneficiar Ministerul Educatiei Nationale, CNCSU), director de proiect M.Frentiu;(1994-1997)
(2) Modelarea si implementarea unei baze multidisciplinare de algoritmi pentru crearea unui centru de calcul de înalta performanta, director de proiect Prof. M.Frentiu (1998-2001)
(3) Cercetari asupra anumitor structuri si metode matematice, cu aplicatii în optimizare, mecanica si informatica, 1979-1981, CNST.

(4) Cercetari privind unele structuri algebrice si metode numerice cu aplicatii în optimizare, mecanica si informatica, 1982-1985, cu MEI.
(5) Contract nr.80/531/1989: Noi algoritmi de clasificare, simulare, reorganizarea datelor, sau pentru rezolvarea unor clase generale de probleme din diferite domenii, în 1989-93.
(6) Cercetari moderne în Informatica: Transformarea programelor si a expresiilor secventiale în programe si expresii paralelizabile, Cu CNST, 1989

Coordinations of international programmes

(1) Coordonator la proiectul Tempus: Advanced Educational Technology Center - Tempus S-JEP 12518-97, 1997-2000

Participations in research or production contracts with industry

(1) Sistem informatic privind urmarirea bolilor profesionale la angajatii din Întreprinderea miniera Cluj-Napoca, 1977-1978.
(2) Sistem informatic privind evidenta produselor, lucrarilor si serviciilor facturate, Întreprinderea Electrometal, Cluj-Napoca, 1978.
(3) Asistenta tehnica pentru perforare, programare si medie mecanizare, Cons. orasenesc Gheorgheni, jud. Harghita, 1977-1978.
(4) Asistenta tehnica pentru perforare, programare si medie mecanizare. Cons. popular jud. Harghita, 1979.
(5) Biblioteca matematica pentru calculatoarele din seria 8000. ITC, 1980 1981.
(6) Elaborarea unui sistem de codificare specifica întreprinderii Electrometal Cluj Napoca, 1982-1983.
(7) Studii privind simularea automatelor programabile, CUG Cluj Napoca, 1985.
(8) Elaborarea unei baze de date pentru subsistemul personal. Beneficiar, Întreprinderea Electrometal, Cluj Napoca, 1987.
(9) Elaborarea bibliotecilor MATH I si MATH II pentru calculatoarele personale, cu ITCI, 1987-1988.
(10) Elaborarea si implementarea unui mediu de programare PASCAL sub sistemul de operare "U". ITCI, 1988.

Babes-Bolyai University of Cluj-Napoca, Faculty of Mathematics and Computer Science, Department of Computer Science, M. Kogalniceanu Str. 1, Cluj-Napoca, ROMANIA
E-mail address: leon@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume L, Number 1, 2005

CORRECTNESS: A VERY IMPORTANT QUALITY FACTOR IN PROGRAMMING

MILITON FRENŢIU

Abstract. Correctness is one of the most important property of a program. Nevertherless, the students are usually not taught to prove the correctness of their programs, and when it is done, they dislike this activity. The importance of program correctness, and the necessity to teach it from the first course in programming, are underlined. Also, some consequences on the education activity are considered.

Keywords: Program correctness, Education, Software Engineering, Stepwise refinement

1. Introduction

Todays software systems have become essential parts of our everyday life. Computers are used in all fields of human activities. More and more programs are needed, and their complexity increases continuously. Software critical systems [But94] require error-free programming. To obtain programs without errors we need a new way of writing programs, new people, much better educated, able to do this.

There is a contradiction between the desire to obtain a system as quickly as possible, and to build a correct system. The experience shows that more than 75% of finished software products have errors during maintenance, and deadlines are missed and cost overruns is a rule not an exception [Gib94]. It was estimated [You90] that more than 50% of the development effort was spent on testing and debugging. Nevertheless, some errors are not detected by testing, and some of them are never detected. More, there are projects that have never been finished [Eff94]. And it is not an exception; it is estimated that from each six large projects two of them are never finished [Gib94, Rob98, or 31.1% according to Sta95]. Also, some other examples and problems may be found in [Gib94].

Due to unreliable software, many people died, and serious economical damages have been produced. There are many well known examples. Some of them are mentioned below, and more (107) can be found in [Der05].

Received by the editors: April 15, 2005.

During the Gulf war, on February 25-th 1991, a Patriot missile failed to intercept an incoming Iraqi Scud missile, due to accumulating errors with real numbers computations [Arn96, Der05]. Due to this error 28 American soldiers died.

On June 4, 1996, the first flight of the Ariane 5 rocket ended in failure. Approximately 40 seconds after initiation of the flight sequence, Ariane 5 exploded [Jez97]. The European Space Agency and French National Center for Space Studies, established an inquiry board to determine the cause of the accident. This accident happened due to a software reuse error of an integer conversion procedure.

On September 23, 1999 the Mars Climate Orbiter was lost when it entered the Martian atmosphere in a lower than expected trajectory [Lev01]. The cause of the accident was in giving data in English units instead of metric units.

According to Andrews [And02], "in the past 20 years, there have been approximately 1100 computer-related accidental deaths.

Let us consider that a program is composed of n procedures, and the probability of correctness of each procedure is at most p. Then the probability of correctness of the entire program is at most p^n [Dah72]. For $p < 1$ and n very large this value is very closed to 0. It means that the only chance to obtain a reliable program is $p = 1$, i.e. each procedure must be perfect. This means that we must prove the correctness of all used procedures, or, as will be shown later, we must obtain a correct procedure when we conceive it.

Therefore, the most important property of a program is whether it accomplishes the intentions of its user, i.e. if it is correct.

2. A short overview on program correctness

The concept of program correctness was introduced by Floyd [Flo67]. Also, a method for proving program correctness was given in the same paper. Before, the correctness of individual algorithms was proved by Hoare [Hoa67, Hoa68, Hoa71], Foley and Hoare [Fol71], London [Lon70a, Lon70b], Naur [Nau66].

In a program P we distinguish three types of variabiles, grouped as three vectors X, Y, and Z. The input vector $X = (x_1, x_2, \ldots, x_m)$ consists of the input variables. They denote the known data of the problem PP solved by the program P. We may suppose they do not change during computation. The output vector $Z = (z_1, z_2, \ldots, z_m)$ consists of those variables which denote the results of the problem PP. The program vector Y consists of the auxiliary variables, which denotes various partially results of the computation.

Two predicates are associated to the program P: an input predicate and an output predicate. The input predicate $\varphi(X)$ is TRUE for those values a of X for each the problem may be solved. The output predicate $\psi(X, Z)$ shows the relation between the results Z and the input values X. It is TRUE for those values a and b of the vectors X and Z for which the results of the problem are b when the initial/input data is a. The specification of the program P is the pair formed from the input predicate $\varphi(X)$ and the output predicate $\psi(X, Y)$.

The program P terminates in respect to the input predicate $\varphi(X)$ if for each value $a = (a_1, a_2, \ldots, a_n)$ of the vector X for which the predicate φ is TRUE, the execution of P terminates. In this case, the computation done by P is the sequence of states passed during the execution, and the value b of the vector Z in the final state is the result of the execution. We may write $b = P(a)$, i.e. P implements a function.

The program P is partially correct with respect to the specification if for the value a for which $\varphi(a)$ is TRUE and the execution terminates with the results $b = P(a)$ then $\psi(a, b)$ is TRUE. The program P is totally correct with respect to $\varphi(X)$ and $\psi(X, Y)$ if the program P terminates with respect to $\varphi(X)$ and it is partially correct with respect to $\varphi(X)$ and $\psi(X, Y)$.

A method for proving partial correctness of a flowchart program is due to Floyd [Flo67] and it uses a set of cut-points. This is a set of points on the arcs of the flowchart such that every loop includes at least one such cut-point. Also, there is a cut-point on the arc leading from the START box, and there is a cut-point on the arc leading to the HALT box.

To each cut-point i of the flowchart, a predicate $\mu_i(X, Y)$ is associated. This predicate, called an invariant predicate, is invariantly true for the current values of X and Y in this cut-point, i.e. it characterizes the relation that must exist between variables at this point. At the START cut-point the corresponding invariant predicate is $\varphi(X)$, and at the HALT cut-point it is $\psi(X, Z)$.

The set of cut-points defines the paths that must be verified. Let α be a path leading from the cut-point i to the cut-point j, with no intermediate cut-points (there can be more such paths). To this path we associate a predicate $R_\alpha(X, Y)$ which gives the condition for the path α to be traversed, and a function $r_\alpha(X, Y)$ such that if Y are the intermediate values in the cut-point i then, when the path is traversed, $Y' = r_\alpha(X, Y)$ are the values of Y in the cut-point j. A verification condition is associated to the path α. This condition is:

$$\forall X \forall Y (\mu_i(X, Y) \wedge R_\alpha(X, Y) \rightarrow \mu_j(X, r_\alpha(X, Y)))$$

Floyd [Flo67] proved that if all the verification conditions are true then the program is partially correct with respect to $\varphi(X)$ and $\psi(X, Z)$.

Floyd also suggested a method for proving termination using well-founded sets. A well-founded set M is a partially ordered set, without infinite decreasing sequences. For each path α from i to j, a termination condition is formed:

$$\varphi(X) \wedge R_?(X, Y) \rightarrow (u_i(X, Y) > u_j(X, r_\alpha(X, Y))$$

Here

$$u_i : D_X \times D_Y \rightarrow M$$

is a function associated to the cut-point i.

If all termination conditions are proved then the program P terminates over $\varphi(X)$.

The ideas of Floyd were developed by Hoare [Hoa69] who introduced an axiomatic method for proving the partial correctness of a program.

Then Dijkstra [Dij75] introduces the important concept of weakest precondition. His idea, of formally deriving correct programs from specifications, was continued by Gries [Gri81] who states that is more important to develop correct programs than to prove latter their correctness: A program and its proof should be developed hand-in-hand with the former usually leading the way. This idea was developed further by Dromey [Dro89], and Morgan [Mor90].

3. Consequences of Program correctness theory on teaching Programming

One way to change the software engineering situation shortly described in the Introduction, also known as "software crisis, is a better education of the new generations of programmers. It is the time to teach programming in conformity with the theory of program correctness. As Naur has underlined in [Nau66], "it is a deplorable consequence of the lack of influence of mathematical thinking on the way in which computer programming is being pursued. We think that teaching programming well is an important part of our tasks as teachers in the universities.

It is not only possible, but necessary, to explicitly teach the methods and principles for good programming. Some early papers on program correctness [Nau66, Lon70a, Lon70b] have proved the correctness of some concrete algorithms. Just Hoare has made a significant move from a posteriori proof of an existing program [Hoa67, Hoa68, Fol71] to a program design method [Hoa71].

We need to teach program correctness for many reasons. First one is the impact this theory has on the future programmers in general. Second, we need very skilled people for program verification activities. The old testing is needed, but not sufficient and not efficient. Program inspection [Gil93] is required for CMM level3 [Pau93], and the inspection team must contain people aware of program correctness theory. More, we need skilled people to use Formal Methods for building all future safety-critical systems [But94].

Another reason for teaching program correctness comes from the consequences of program correctness on programming methodology. Why do we need specifications? What is the importance of program clarity and simplicity in software engineering? Why we must write all kind of documents?

Some of the most important rules considered important for programming well, which are consequences of the program correctness theory, are given in [Fre93, Fre94, Fre97]. We select some of the most important and simple ones below (specifying the original bibliographical source):

- Define the problem completely (i.e., write the precondition $\varphi(X)$ and the postcondition $\psi(X, Z)$) [Gri81, Led75, Sch90].
- Think first, program later [Led75].

- Write and use modules as much as possible [Led75, Sch90].
- Prove the correctness of algorithms during their design [Gri81].
- Decide which are the needed program variables, and what are their meanings. Write invariants for these variables and insert them as comments in the program [Gri81, Led75].
- Choose suitable and meaningful names for variables [Led75].
- For each variable of a program, make sure that it is declared, initialised and properly used [Nau66].
- Verify the value of a variable immediately it was obtained [Nau66].
- Use comments to document the program [Led75].
- Verify each part of a program as soon as possible [Gri81, Sch90, Gil93].
- Use symbolic names for all entities (constants, types, variables, procedures and functions) [Fre94].
- Avoid to use global variables [Sch90].
- Hand-check the program before running it [Led75].
- Write the documentation of the program simultaneously with its building [Sch90].
- Give attention to the clarity and simplicity of your program [Fre01a]!

Some people [Ste91] are against proving correctness. Others consider it is expensive, since they think the effort to build a program in such a way is considerable increased.

Certainly, proving correctness of a real large program is too complicated and, maybe, inappropriate. But the correctness of the important and difficult procedures used in the system may be proved. And the specifications of these procedures are not changed from time to time by the client, as some "researchers argue against proving correctness.

Also, proving correctness has an important impact on program verification. It is well known that proving correctness and testing complete each other. There are some aspects (performance, for example) that can be verified only by testing. But testing is a time consuming activity that must be reduced through other forms of verifications (inspections [Fag76, Mye78, Gil93], symbolic executions [Kin76]). And a correct algorithm is not accompanied by debugging. When testing discover errors we must start the most difficult and unpleasant life-cycle activity, which is debugging. The minor errors may easily be corrected, meanwhile debugging logical errors consumes much time and effort, and often are unsuccessful.

Almost all students have learned some programming at school. They think they know what programming is all about. They express resentment when they a forced to document their activity, to design the program, or to think to the correctness of their programs. They run directly to computer and introduce their programs, and run them. If the first execution seems OK they are satisfied. They are not used to test seriously their programs. We need to fight with these people, to change their habit, to educate them in a different way. That is why we have decided to

stress from the very beginning that "programming is a high intellectual activity [Hoa91], that the correctness is the most important property of good programs.

According to ISO-9126 the factors of the program quality are functionality, reliability, usability, efficiency, maintainability, and portability. Is program correctness present in defining the quality of a program? We put this question since it is not directly mentioned in the quality factors.

Nevertheless, correctness is clearly present in the above mentioned factors. Functionality supposes the program is correct, and reliability expects as few errors as possible. And maintainability is increased in many ways if the program is correct. First, crorrective maintenance is not needed for a correct program. Second, perfective and adaptive maintenance is easier for a well design program. And correctness is strongly connected to good design.

The software engineering course "Algorithms and Pascal programming was given to the first year computer science students. The course concerned was presented during their first semester in the University. It is the first course on programming, and the main parts of it are: a Pseudocode language for describing algorithms, developing correct algorithms from specifications, simple Pascal programming, the life-cycle of a program, the methods of designing, coding and verifying simple programs.

There are some years since the notion of correctness and the Floyds method for proving correctness are taught in this course. Also, it was underlined that it is more important to construct a correct program than to prove later its incorrectness. But is for the first time when the accent was given to develop correct algorithms from specifications, not on proving correctness.

We used for many years a Pseudocode language to describe algorithms. This language has those three computation structures needed to write structured algorithms: the sequential structure, the alternative and the iterative computation structure. Each year stepwise refinement [Wir71] was considered a very important programming method, but it was presented in an informal way. In 2004, for the first time, the development of correct algorithms from specifications was used in a more rigorous way [Mor90]. The refinement rules for assignement, for the sequential composition, alternation computation structure (if-then-else), and for iteration were adapted to the used Pseudocode language [Fre04].

We gave many examples of refinement, and the students exercised these rules. At the final examination we observed an improvement in the correctness of student algorithms. The results at students exams in 2004 were compared to those of the students on 2003 [Fre04], and the statistical hypothesis on mean values equality was verified. It was rejected, since the results were significantly better in 2004. Therefore, we may conclude that the correctness of the students algorithms at the examination has improved in the last year. Certainly, their may be more reasons for this, but the main difference consisted only in the introduction of the development of algorithms from specifications, using clear defined rules.

4. Conclusions

We need confidence in the quality of our software products. We need to educate the future software developers in the spirit of producing correct, reliable systems. For this we must teach students to develop correct programs. We are aware that usually programmers do not prove the correctness of their pograms. There always must be a balance between cost and the importance of reliability of the programs. But just when the well educated people do not prove the correctness, their products are more reliable than the products of those "programmers who never studied program correctness. Therefore, we consider that the students must hear, and must pay attention to the correctness of their products.

We think that we educate our students for their future profession, that last for several decades. Also, they must be prepared for changes in software engineering. They need to acquire now the knowledge needed to build more reliable systems. Our simple experiment confirmed the improvement in the correctness of students algorithms compared to the previous year.

Certainly, proving correctness of algorithms is not enough to obtain more reliable systems, but it is necessary. For years we ask the students to understand and to respect some important rules of programming [Fre93, Fre00, Fre03, Led75]. Many of them are simple consequences of the theory of program correctness [Fre97].

Thus, from first year, students are taught about program specification and design. The entire life-cycle is presented, the importance of documentation for all steps is underlined. Top-down and the other programming methods are taught, and the students hear that the design is more important than coding. Each part of the design must be specified, the code must be explained by comments, the comments should be neither more nor less than needed. Since we also observed that students do not like to write comments [Fre02] we tried to explain their necessity, and to force them to document the code [Fre03].

The fact that program verification is a very important activity which extends from the first statement of the problem, until the end of the project is repeated many times. And a special attention was given to the inspection of all documents. The students must hear from the beginning that testing is not enough, that inspection of all phases may be more useful. Thats why we ask the students to have a notebook at their laboratories, and the entire life-cycle of their programs must be reflected in this notebook. And they must respect the order of the activities; the specification and the design must be before coding.

We must fight against the idea that proving correctness is expensive, that proving needs time the developers think they do not have. We must tell them they have very much time for debugging!

References

(And02) Andrews P., Safety-critical projects: Can formal methods help?, Builder-com, 2002, sept. 12.

(Arn96) Arnold D.N., Two disasters caused by computer arithmetic errors, http://www.ima.umn.edu/∼arnold/455.f96/disasters.html

(But94) Butler, R.W., S.C.Johnson, Formal Methods for Life-Critical Software, NASA Langley Research Center, http://smesh.larc.nasa.gov/

(Dah72) Dahl O.J., E.W.Dijkstra, and C.A.R.Hoare, Structured programming, Academic Press, New York, 1972.

(Der05) Dershowitz N., Software horror stories, School of Computer Science, TelAviv University, http://www.cs.tau.ac.il/∼nachumd/verified/horror.html

(Dij75) Dijkstra, Guarded commands, nondeterminacy and formal derivation of programs, Comm.A.C.M., 18(1975), 8, pp.453-457.

(Dro89) Dromey G., Program Derivation. The Development of Programs from Specifications, Addison Wesley, 1989.

(Eff94) Effy Oz, When Professional Standards are LAX. The CONFIRM Failure and its lessons, Comm. A.C.M., 37(1994), 10, 29-36.

(Fag76) M. Fagan, Design and Code Inspections to Reduce Errors in Program Development, IBM Systems Journal, 15 (3), 1976.

(Flo67) Floyd R.W., Assigning meanings to programs, Proc. Symposium in Applied Mathematics, 19, AMS, 1967, pp.19-32.

(Fol71) Foley M., and C.A.R.Hoare, Proof of a recursive program: Quicksort, The Computer Journal, vol.14,n.4, p.391-395.

(Fre84) Frentiu M., On the program correctness, Seminar on Computer Science, preprint 1984, 75-84.

(Fre93) M.Frentiu, B.Pârv, Programming Proverbs Revisited, Studia Universitatis Babes-Bolyai, Mathematica, XXXVIII (1993), 3, 49-58.

(Fre94) M.Frentiu, B.Pârv, Elaborarea programelor. Metode si tehnici moderne, Ed.Promedia, Cluj-Napoca 1994, 221 pages, ISBN 973-96114-9-4.

(Fre95) M.Frentiu, Reguli de programare pentru incepatori, in Lucrarile Conferintei "Informatizarea invatamantului, Balti, 4-7 octombrie 1995.

(Fre97) M.Frentiu, On program correctness and teaching programming, Computer Science Journal of Moldova, vol.5(1997), no.3, 250-260.

(Fre00) M.Frentiu, On programming style program correctness relation, Studia Univ. Babes-Bolyai, Seria Informatica, XLV(2000), no.2, 60-66.

(Fre01a) M.Frentiu, Verificarea Corectitudinii Programelor, Univ. "Petru-Maior", Tg. Mures, 2001, 116 pagini, ISBN 973-8084-32-6.

(Fre01b) M. Frentiu, and H.F.Pop, A Study of License Examination Results Using Fuzzy Clustering Techniques, Research Seminar on Computer Science, 2001, pp. 99-106.

(Fre02a) M. Frentiu, The Impact of Style on Program Comprehensibility, Proceedings of the Symposium Zilele Academice Clujene, 2002, pp. 7-12.

(Fre02b) M. Frentiu, H.F.Pop, A Study of Dependence of Software Attributes using Data Analysis Techniques, Studia Universitatis Babes-Bolyai, Informatica, 47(2),2002, 53-60.

(Fre03) M. Frentiu, On programming style, Babes-Bolyai University, Department of Computer Science, http://www.cs.ubbcluj.ro/∼mfrentiu/articole /style.html

(Fre04a) M. Frentiu, Program Correctness in Software Engineering Education, Proceedings of the International Conference on Computers and Communications ICCCM4, pp.154-157, Oradea, 2004, may 27-29.

(Fre04b) M. Frentiu, Formal Methods in Software Engineering Education, Proceedings ICELM1, Tg. Mures, 2004.

(Fre04c) Frentiu M., An Overview on Program Inspection, Proceedings of the Symposium "Zilele Academice Clujene", 2004, pp. 9-14.

(Gib94) Gibs W.W., Softwares Chronic Crisis, Scientific American, september, 1994.

(Gil93) Tom Gilb and Dorothy Graham, Software Inspection, Addison-Wesley, 1993.

(Gri81) Gries D., The Science of Programming, Springer-Verlag, Berlin, 1981.

(Hoa69) Hoare C.A.R., An axiomatic approach to computer programming, Comm. A.C.M., 12 (1969), pp. 576-580.

(Hoa71) C.A.R.Hoare, Proof of a program: FIND, Comm.ACM, 14(1971), pp.39-45.

(Hoa91) Hoare C.A.R., The Mathematics of Programming, LNCS, 206, 1991, pp.1-18.

(Iga75) Igarashi S., London R.L., and Luckham D.C., Automatic Program Verification I: a logical basis and its implementation, Acta Informatica, 4(1975), 145-182.

(Jez97) Jezequel J.M., B.Meyer, Put it in the contract. The lessons of Ariane, Computer (IEEE), vol.30 (1997), no.2, p.129-130.

(Kat76) Katz, and Manna, Logical analysis of programs, Comm.ACM, 19(1976), 4, p. 188-206.

(Kin76) King J.C., Symbolic Execution and Program Testing, Comm. ACM, 19 (1976), 7, p.385-394.

(Led75) Ledgard H.F., Programming Proverbs for Fortran Programers, Hayden Book Company, Inc., New Jersey, 1975.

(Lev01) Leveson N.G., Systemic Factors in Software-Related Spacecraft Accidents, AIAA 2001, 47-63.

(Lon70a) London R.L., Proving Programs Correctness. Some Techniques and Examples, BIT, 10 (1970), pg.168-182.

(Lon70b) London R.L., Proof of Algorithms. A new kind of cerification, Comm. ACM, 13 (1970), pg.371-373.
(Mor90) Morgan, C., Programming from specifications, Springer, 1990.
(Mye78) Myers, A., A Controlled Experiment in Program Testing and Code Walkthroughs Inspection, Comm.A.C.M., 21(1978), no.9, pp.760-768.
(Nau66) Naur, Proof of Algorithms by general snapshots, BIT, 6(1966), pg.310-316.
(Nic04) V.Niculescu, M. Frentiu, Designing Correct Parallel Programs from Specifications, Proceedings of the 8th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2004), Orlando, USA, July 18-21, 2004, pp.173-178.
(Pau93) Paulk M.C., B.Curtis, M.B.Chrissis, C.V.Weber, The Capability Maturity Model for Software, Tech.Report, CMU/SEI-93-TR-25, and IEEE Software, 10(1993,4, 18-27.
(Rob98) Robinson H., et all, Postmodern Software Development, The Computer Journal, 41(1998), 6, p.363-375.
(Sch90) Schach S.R., Software Engineering, IRWIN, Boston, 1990.
(Sta95) The Standish Group Report: Chaos, http://www.scs.carleton.ca/∼bean /PM/Standish-Report.html
(Ste91) Stevenson D.E., 1001 Reasons for not Proving Program Correct: A Survey, Philosophy and Computers, 1, 1991.
(You90) Yourdon, E., Modern SoftwareAnalysis, Yourdon Press, Prentice Hall Buiding, New Jersey 07632, 1989
(Weg74) Wegbreit, The synthesis of loop predicates, CACM, 17(1974), 102-112.
(Wir71) Wirth N., Program development by stepwise refinement, Comm. ACM 14(1971), 4, 221-227.

Babes-Bolyai University of Cluj-Napoca, Faculty of Mathematics and Computer Science, Department of Computer Science, M. Kogalniceanu Str. 1, Cluj-Napoca, ROMANIA

E-mail address: mfrentiu@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume L, Number 1, 2005

CONTINUATIONS FOR REMOTE OBJECTS CONTROL

ENEIA TODORAN[1], FLORIAN MIRCEA BOIAN[2], CORNELIA MELENTI[1], AND NIKOLAOS PAPASPYROU[3]

Abstract. We have recently introduced the "continuation semantics for concurrency" (CSC) technique in an attempt to exploit the benefits of using continuations in concurrent systems development. CSC is a general technique for denotational semantics which provides excellent flexibility in the compositional modeling of concurrent control concepts. In this paper, we present a denotational semantics designed with CSC for a distributed languages incorporating two control concepts which have not been modeled denotationally before: remote object (process) destruction, and cloning.

1. Introduction

The CSC technique was recently introduced by us [13, 14] in an attempt to exploit the benefits of using continuations in concurrent languages development. It is a general technique for denotational semantic design, which can be used to model both sequential and parallel composition in interleaving semantics, as well as various mechanisms for synchronous and asynchronous communication [13]. Intuitively, it is a semantic formalization of a process scheduler simulated on a sequential machine. In the CSC approach, a *continuation* is an application-dependent configuration (structure) of computations, where by *computation* we mean a partially evaluated denotation (meaning function). Every moment there is only one *active* computation, which remains active only until it performs an elementary action. Subsequently, another computation taken from the continuation is planned for execution. In this way it is possible to obtain the desired interleaving behavior for parallel composition.

A continuation is a representation of what remains to be computed after taking an elementary step from the (currently) active computation. This corresponds to the original definition of continuations[1], but in the CSC approach continuations are structured entities and each computation contained in a continuation can be accessed and manipulated separately. Synchronization and communication information can also be encoded in continuations. What one gets with CSC is a

Received by the editors: January 10, 2005.

[1]A continuation is a representation of the rest of the computation, according to [11].

"pure" continuation based approach to communication and concurrency in which all control concepts are modeled as operations manipulating continuations.

As shown in [15], by using the CSC technique, denotational semantics can be used not only for formal specification and design, but also as a general method for building compositional interpreters for concurrent programming languages. In this approach, a denotational (compositional) mapping can use continuations for concurrency to produce incrementally a single stream of observables, i.e. a single execution trace, rather than an element of a power domain construction. By using a random number generator an arbitrary execution trace is chosen, thus simulating the non-deterministic behavior of a "real" concurrent system. Following [15], we will call such a compositional mapping a *denotational prototype*.

In this paper, we employ the CSC technique in designing a denotational semantics and a corresponding denotational prototype for a simple distributed language providing operations for remote object (process) control. In the sequel, an *object* is a thread (sequence) of computations with a local state. Distributed states are essential for defining the semantics of concurrent languages used in distributed computing. The language that we study in this paper provides a mechanism for synchronous communication taken from CSP [4]. It incorporates a notion of remote processes as *named objects* and allow object-to-object communication, as well as remote object destruction and cloning.

The last two operations can be encountered at operating system level, in some coordination languages [5], or in distributed object oriented and multi-agent systems such as Obliq and IBM Java Aglets [3, 7, 6]. The former operation kills a parallel running object and is similar to the "`kill -9`" system call in Unix. The latter operation creates an identical copy of a (parallel) running object. In this paper, we provide an accurate denotational semantic model for remote object destruction and cloning. To the best of our knowledge, such operations for remote object control have not been modeled denotationally until now, and all our attempts solve the problem by using only classic compositional techniques have failed.

Instead of using mathematical notation for the definition of the denotational models, we use the (lazy) functional programming language Haskell [9]. In this way, as in [14], we avoid unnecessary complexities accompanying the use of domain theory or the theory of metric spaces, which could have been adopted alternatively. At the same time, we allow our denotational models to be directly implementable, in the form of interpreters for the language under study, and thus to be easily tested and evaluated. The denotational semantics will only be tested on trivial (non-recursive) example programs. However, the corresponding denotational prototype will be tested on "real-life" examples. For example, we present a simple concurrent generator of prime numbers based on the sieve of Erathostenes.

2. Syntax and informal explanation

We consider a simple distributed language called L_{obj}. The syntax of L_{obj} is given below in BNF. The basic components are a set $(v \in)V$[2] of *data variables*, a set $(n \in)N$ of *numerical expressions*, a set $(b \in)B$ of *boolean expressions*, and a set $(y \in)Y$ of *procedure variables*. The language also uses a class $(w \in)W$ of *object variables*; while a data variable holds a data value (in our case an integer value) an object variable holds an object reference. L_{obj} comprises a simple language of expressions, supporting basic operators on numerical values and boolean values. In the grammar that follows, z denotes an integer constant, and v denotes a (numerical) variable.

$$\begin{array}{lll} n & ::= & z \mid v \mid n + n \mid n - n \mid n \,\%\, n \mid \ldots \\ b & ::= & n == n \mid n < n \mid \ldots \end{array}$$

L_{obj} provides assignment ($v := n$), a primitive for writing the value of a numerical value (i.e. for producing an intermediate observable) at the standard output file (**write** n)[3], a null command (**skip**), recursion, a conditional command (**if** b **then** x **else** x), sequential composition ($x\,;\,x$), guarded nondeterministic choice (**ned** $[(\gamma \rightarrow x)^*]$), together with constructs for object (process) creation (**new** w **is** x), destruction (**kill** w), and cloning (**clone** w **is** w). The syntax of L_{obj} is formally defined as follows:

$$\begin{array}{lll} x & ::= & \textbf{skip} \mid v := n \mid \textbf{write}\ n \mid \textbf{call}\ y \mid \textbf{letrec}\ y\ \textbf{be}\ x\ \textbf{in}\ x \\ & \mid & \textbf{if}\ b\ \textbf{then}\ x\ \textbf{else}\ x \mid \textbf{ned}\ [(\gamma \rightarrow x)^*] \mid x\,;\,x \\ & \mid & \textbf{new}\ w\ \textbf{is}\ x \mid \textbf{kill}\ w \mid \textbf{clone}\ w\ \textbf{is}\ w \end{array}$$

where

$$\gamma \ ::= \ w\,!\,n \mid ?\,v$$

The guards γ of a non-deterministic choice are constructs for *object-to-object* synchronous communication. In L_{obj}, an *object* is a thread (sequence) of computations acting on a *local state*. There is no shared memory area. Parallel objects can only communicate by exchanging messages. The communication mechanism is taken from CSP [4]. A communication can take place by the synchronous execution of two actions $w\,!\,n$ and $?\,v$, occurring in parallel objects. The primitive $w\,!\,n$ evaluates the expression n and sends the value to the object referred by w. An object executing the $?\,v$ statement is willing to communicate with an arbitrary partner object, as long as that partner explicitly mentions the name of the object in which the $?\,v$ primitive occurs; upon synchronization, the primitive $?\,v$ assigns the received value to the variable v. The expression n is evaluated in

[2] In this paper, the notation $(x, y, ...)X$ introduces the set X with typical variables x, y,...

[3] Expressions of boolean type can not be assigned or output; they can only be used as conditions.

the memory area of the sending object and the result is assigned to the variable v in the memory area of the receiving object. In order to communicate, two parallel objects synchronize (the first one that is ready to communicate waits for the other) and then they exchange a single value.

The **new** w **is** x statement can be used to create a new object (with a new private state) which evaluates x. A reference to the newly created object is assigned to variable w. Therefore, this statement not only creates a new object but also a new communication connection to this object, which can be used by a $w\,!\,n$ primitive. Moreover, the new reference can be used by the two constructs for remote object control: **kill** w and **clone** w_1 **is** w . The former destroys the parallel running object to which variable w refers. The latter clones the parallel running object referred by w and assigns the clone's identifier to the object variable w_1.

In Haskell, we implement the syntax of L_{obj} as follows:

```
type V = String
type W = String
type Y = String
data N = Z Int | V V | Plus N N | Minus N N | Mod N N
data B = Eq N N | Lt N N
data C = Snd W N | Rcv V
data X = Skip | Assign V N | Write N | Call Y  | LetRec Y X X
       | If B X X | Ned [(C,X)] | Seq X X
       | New W X | Kill W | Clone W W
```

3. Denotational semantics

L_{obj} is a language with distributed objects. Objects can be referred and controlled by using *object identifiers* (or *references*). For simplicity, we represent object references by integer numbers.

```
type O = Int
```

Each object in L_{obj} has a *local state*, which can be accessed or modified in an imperative manner. A state is usually represented as a mapping from variables to values. In L_{obj} a state has two components: one for data values and the other one for object references. We implement states by the type `S`. The operations `getv`, `setv`, `getw` and `setw` provide the basic functionality of a state.

```
type S = (W -> O,V -> Int)

getv :: V -> S -> Int
getv v (sw,sv) = sv v
```

```
setv :: S -> V -> Int -> S
setv (sw,sv) v i = (sw,subs sv v i)

getw :: W -> S -> O
getw w (sw,sv) = sw w

setw :: S -> W -> O -> S
setw (sw,sv) w o' = (subs sw w o',sv)
```

The mapping `subs` is defined as follows:

```
subs :: (Eq a) => (a -> b) -> a -> b -> (a -> b)
subs f x y = \x' -> (if (x==x') then y else f x')
```

We can already define simple valuations `evN` and `evB`, for numerical and boolean expressions. In general, the meaning of a (boolean) expression depends on the current state.

```
evN :: N -> S -> Int
evN (Z n) s = n
evN (V v) s = getv v s
evN (Plus n1 n2) s = (evN n1 s) + (evN n2 s)
evN (Minus n1 n2) s = (evN n1 s) - (evN n2 s)
evN (Mod n1 n2) s = (evN n1 s) `mod` (evN n2 s)

evB :: B -> S -> Bool
evB (Eq n1 n2) s = (evN n1 s) == (evN n2 s)
evB (Lt n1 n2) s = (evN n1 s) < (evN n2 s)
```

In L_{obj} it is possible for a program to block, if all parallel objects are waiting at nondeterministic constructs that do not have matching communication primitives. Such a deadlock is fundamentally different from non-termination (e.g. a procedure that repeatedly calls itself) and we expect it to be detected by the denotational semantics. We use the type `Q` to represent streams (lists) of observables. In the definition given below, `Epsilon` denotes *normal termination* and `Deadlock` denotes *deadlock*.

```
data Q = Epsilon | Deadlock | Q Int Q
```

We use the following `Show` instance to visualize the yields of our denotational models.

```
instance Show Q where
   show Epsilon = " "
   show Deadlock = " deadlock "
   show (Q n q) = " " ++ (show n) ++ (show q)
```

The denotational semantics for L_{obj} maps each statement to a *computation* (a partially evaluated denotation), which is an element of type `D`. We will use continuation semantics for concurrency, therefore it is reasonable to assume that a computation is a function that depends on the current continuation. In the definition below, `Cont` is the semantic class of continuations. The semantics of a program also depends on the current state.

```
type D = Cont -> S -> Final
```

`Final` is the final yield of the denotational mapping. In section 4, `Final` will implement a domain for random execution traces [15]. In this section, `Final` implements a classical power domain construction [10], and the denotational semantics produces the collection of all possible traces for any given program; the Haskell definition will be given later.

Following the CSC technique [13, 14], a continuation is a configuration of computations that can be executed in parallel. The CSC technique is very general. It does not impose any restriction on the structure of continuations. For our language with object creation it is convenient to define continuations to be multisets of objects. Objects are elements of type `Obj`. An object is a triple, consisting of an object identifier, a thread (sequence) of computations, and a local state. We use two basic notions to model the flow control: the *stack* to model sequential composition, and the *multiset* to model parallel composition. A stack models a single thread (or sequence) of computations. We implement both stacks and multisets as Haskell's lists. The type `PC` implements a multiset. The type `SC` implements a stack. An element of a `SC` stack is either a computation or a non-deterministic choice consisting of a list of guarded alternatives, where each alternative consists of a (synchronous) communication attempt and a computation. Haskell definitions are as follows:

```
type Cont = PC
type PC = [Obj]
type Obj = (O,SC,S)
type SC = [Comp]
data Comp = D D | S [(SemC,D)]
```

We use some auxiliary mappings on objects.

```
idOf :: Obj -> O
idOf (o,sc,s) = o

threadOf :: Obj -> SC
threadOf (o,sc,s) = sc

stateOf :: Obj -> S
```

```
stateOf (o,sc,s) = s

updThread :: Obj -> SC -> Obj
updThread (o,sc,s) sc' = (o,sc',s)

updState :: Obj -> S -> Obj
updState (o,sc,s) s' = (o,sc,s')
```

The type `SemC` implements communication attempts. The function `semC` maps the (syntactic) communication primitives of L_{obj} to corresponding (semantic) communication attempts.

```
data SemC = SemSnd W (S -> Int) | SemRcv V

semC :: C -> SemC
semC (Snd w e) = SemSnd w (evN e)
semC (Rcv v) = SemRcv v
```

The function `k` implements *continuation completion*. It maps a continuation to the program answer that would result if the continuation alone was left to execute. It first normalizes the continuation by using the auxiliary mapping `re`. The execution terminates if the (normalized) continuation is empty. Otherwise, `k` calls the function `kc` which implements a scheduler, by using the auxiliary functions `schedc`, `comp`, `scheds`, and `send`.

```
k :: Cont -> Final
k c = case (re c) of {
           [] -> epsilon;
           c  -> kc c;
      }

kc :: Cont -> Final
kc c = case ((schedc c) ++ (scheds c [])) of {
            []  -> deadlock;
            scd -> bigned (map exe scd);
       }

schedc :: PC -> [Sched]
schedc pc =
    [ (Schedc d (obj:pc') (stateOf obj)) | (D d,obj:pc') <-
     comp pc [] ]

comp :: PC -> PC -> [(Comp,PC)]
```

```
comp [] pc' = []
comp (obj:pc) pc' =
    (let p:sc = threadOf obj in
     [(p,(updThread obj sc):(pc ++ pc'))])
 ++ (comp pc (obj:pc'))

scheds :: PC -> PC -> [Sched]
scheds [] pc' = []
scheds (obj:pc) pc' =
    (send [obj] (pc ++ pc')) ++ (scheds pc (obj:pc'))

send :: PC -> PC -> [Sched]
send pc1 pc2 =
    [ (Scheds ((addc (D d1) (obj1:pc1')) ++
         (addc (D d2) (updState obj2 (setv (stateOf obj2)
                        v (pe (stateOf obj1))):pc2')))) |
      (S snd,obj1:pc1') <- comp pc1 [],
      (S rcv,obj2:pc2') <- comp pc2 [],
      (SemSnd w pe,d1) <- snd, (SemRcv v,d2) <- rcv,
      (getw w (stateOf obj1)) == idOf obj2
    ]
```

Continuations are multisets of objects. The semantic operators are designed in such a way as to maintain the following *invariant* of the continuations: *the thread of each object in a continuation is always non-empty, with the possible exception of the leftmost one which conceptually contains at its head the* active *computation.* The normalization function `re` removes the leftmost object in a continuation in case its thread has remained empty after taking an elementary step from the active computation.

```
re :: Cont -> Cont
re ((o,[],s):pc) = pc
re pc = pc
```

Both ordinary computations and pairs of communicating processes are handled by the scheduler mapping `kc`. The function `schedc` handles ordinary computations. The function `scheds` handles pairs of communicating objects (processes). The scheduler computes all possible schedules for a given continuation. Deadlock is detected when there are no schedules. A schedule is an element of the type `Sched`.

```
data Sched = Schedc D Cont S | Scheds Cont
```

Schedules of the form `Schedc d c s` are produced by the function `schedc`. Schedules of the form `Scheds c` are produced by `scheds`. The mapping `exe` executes a single schedule.

```
exe :: Sched -> Final
exe (Schedc d c s) = d c s
exe (Scheds c) = k c
```

The scheduler also uses the mapping `bigned` to compute the meaning corresponding to all possible schedules. A possible definition of `bigned` for a classical power domain semantics is given later in this section. An alternative definition of `bigned`, suitable for computing a single arbitrary execution trace, is considered in section 4.

The denotational function for L_{obj} uses the following semantic operators for modeling the flow of control: `addc`, `new`, `kill` and `clone`. The operator `addc` adds a computation to the continuation for sequential composition. The operators `new`, `kill` and `clone` are used for object creation, destruction, and cloning respectively.

```
addc :: Comp -> Cont -> Cont
addc p (obj:pc) = (updThread obj (p:threadOf obj)):pc

new :: Comp -> Cont -> W -> S -> Cont
new p (obj:pc) w s = let on = newo (obj:pc)
      in  (updState obj (setw s w on)):(on,[p],s0):pc

kill :: Cont -> W -> S -> Cont
kill pc w s =
   aux pc (getw w s)
   where aux :: PC -> O -> PC
         aux [] ok = []
         aux (obj:pc) ok =
             if (idOf obj == ok) then pc else (obj:(aux pc ok))

clone :: Cont -> W -> W -> S -> Cont
clone (obj:pc) wn wo s =
   let on = newo (obj:pc)
   in  aux (updState obj (setw s wn on):pc) (getw wo s) on
       where aux :: PC -> O -> O -> PC
             aux [] oo on = error "clone: invalid object name"
             aux (obj:pc) oo on =
                 if (idOf obj == oo) then
                    case (threadOf obj) of {
```

```
                    [] -> obj:pc;
                    _  -> obj:(on,threadOf obj,stateOf obj):pc;
                  }
              else obj:aux pc oo on
```

The mapping `newo` takes as parameter a continuation and creates a new fresh object identifier. It returns an identifier which is not already in use by some object in the given continuation.

```
newo :: Cont -> O
newo c = (maximum [ idOf obj | obj <- c ]) + 1
```

For handling recursion, we use semantic environments and a fixed-point operator. A semantic environment is a mapping from procedure variables to computations.

```
type Env = Y -> D

fix :: (a -> a) -> a
fix f = f (fix f)
```

We are finally prepared to present the denotational semantics for L_{obj}.

```
sem :: X -> Env -> D
sem Skip e c s = k c
sem (Assign v n) e (obj:pc) s  =
    k (updState obj (setv s v (evN n s)):pc)
sem (Write n) e c s = prefix (evN n s) (k c)
sem (Ned gx) e c s =
    k (addc (S [(semC c,sem x e) | (c,x) <- gx ]) c)
sem (Seq x1 x2) e c s = sem x1 e (addc  (D (sem x2 e)) c) s
sem (Call y) e c s = e y c s
sem (LetRec y x1 x2) e c s =
    sem x2 (subs e y (fix (\d -> (sem x1 (subs e y d))))) c s
sem (If b x1 x2) e c s =
    if (evB b s) then (sem x1 e c s) else (sem x2 e c s)
sem (New w x) e c s = k (new (D (sem x e)) c w s)
sem (Kill w) e c s = k (kill c w s)
sem (Clone wn wo) e c s = k (clone c wn wo s)
```

When the CSC technique is employed in semantic design one can use a *linear-time* domain (see [1]) as final yield of a denotational model for synchronous communication [13]. Intuitively, an element of type `Final` is a set of `Q` sequences of observables. The constants `epsilon` and `deadlock` are of the type `Final`. The former models normal termination and the latter models deadlock detection in the `Final` domain. The `bigned` operator computes the union of a list of elements of

type `Final`. To this end, it is convenient to make `Q` an instance of `Eq`. The `prefix` operator implements the prefixing of an observable to a final program answer.

```
type Final = [Q]

instance Eq Q where
   Epsilon == Epsilon = True
   Deadlock == Deadlock = True
   (Q n1 q1) == (Q n2 q2) = (n1 == n2) && (q1 == q2)
   _ == _ = False

epsilon, deadlock :: Final
epsilon = [Epsilon]
deadlock = [Deadlock]

prefix :: Int -> Final -> Final
prefix n p = [ (Q n q) | q <- p ]

bigned :: [Final] -> Final
bigned [] = []
bigned (q:p) = q `union` (bigned p)

union :: (Eq a) => [a] -> [a] -> [a]
union [] ys = ys
union (x:xs) ys =
   if (x `elem` ys) then (xs `union` ys) else x:(xs `union` ys)
```

In order to test our denotational semantics we define *initial* values for the semantic environment, continuation, and state.

```
e0 :: Env;    e0 y c s = epsilon;
c0 :: Cont;   c0 = [(o0,[],s0)];
o0 :: O;      o0 = 0;
s0 :: S;      s0 = (\w -> o0,\v -> 0);
```

For experiments, we consider the following example programs in L_{obj}.

```
x1 = Seq (New "w1" (Write (Z 1))) (Seq (New "w2" (Write (Z 2)))
 (Write (Z 3)))
x2 = Seq (New "w1" (Ned [(Rcv "v",Write (V "v"))]))
         (Ned [(Snd "w1" (Z 1),Ned [(Rcv "v",Skip)]),
               (Snd "w1" (Z 2),Write (Z 2))])
x3 = Seq (New "w1" (Ned [(Rcv "v",
  Seq (Write (Z 1))
```

```
                              (Seq (Write (Z 2))
                                   (Ned [(Rcv "v",Skip)])))]))
          (Ned [(Snd "w1" (Z 0),
           Seq (Clone "w2" "w1")
                     (Seq (Clone "w3" "w1")
                          (Seq (Kill "w1")
                          (Seq (Kill "w2") (Kill "w3"))))])
```

One can perform the following tests[4]:

```
Main> sem x1 e0 c0 s0
[ 3 2 1 , 3 1 2 , 2 1 3 , 2 3 1 , 1 3 2 , 1 2 3 ]
Main> sem x2 e0 c0 s0
[ 1 deadlock , 2 2 ]
Main> sem x3 e0 c0 s0
[ , 1 2 2 1 2 , 1 2 2 1 , 1 2 1 2 2 , 1 1 2 2 2 , 1 1 2 1 , 1 1 2
1 2 , 1 1 2 1 2 2 , 1 1 2 2 1 2 , 1 1 2 2 1 , 1 1 1 , 1 1 1 2 , 1
1 1 2 2 , 1 1 1 2 2 2 , 1 1 , 1 1 2 2 , 1 1 2 , 1 2 1 1 2 2 , 1 2
1 1 , 1 2 1 1 2 , 1 2 1 2 1 , 1 2 1 2 1 2 , 1 2 1 2 , 1 2 1 , 1 ,
1 2 2 2 , 1 2 , 1 2 2 ]
```

4. Denotational prototype

In [15] we have introduced the notion of a *denotational prototype*. A denotational prototype is a compositional mapping that produces a single execution trace for a given concurrent program rather than the collection of all possible traces. By using a random number generator, an arbitrary execution trace is chosen, thus simulating the non-deterministic behavior of a "real" concurrent system. A denotational prototype is a compositional interpreter for the concurrent language under study, which can be used without difficulty to test non-trivial concurrent algorithms.

It is very easy to modify the denotational semantics given in section 3 to get a denotational prototype for L_{obj}. We change the definition of the type `Final` to reflect the fact the the final yield of the denotational prototype is a sequence of observables (a single execution trace) of type `Q`. The denotational prototype simulates the selection of an arbitrary execution trace by using a random number generator, which is an element of type `RNG`.

```
type Final = RNG -> Q
```

The new definitions for `epsilon` and `deadlock` are as follows:

[4]We accomplished the experiments by using the Hugs interpreter available from `http://www.haskell.org`.

```
epsilon, deadlock :: Final
epsilon = \rng -> Epsilon
deadlock = \rng -> Deadlock
```

Random numbers are natural numbers. A random number generator is a pair consisting of a random number and a mapping that produces a new random number from a given one. `rng0` is a poor man's random number generator that will be used to test our denotational prototype for L_{obj}.

```
type R = Int

type RNG = (R,R -> R)

rng0 :: RNG
rng0 = (17489,\r -> ((25173*r+13849) `mod` 65536))
```

All that remains to be done is to adapt the definitions of `bigned` and `prefix` to deal with single arbitrary execution traces. The new definitions are given below.

```
bigned :: [Final] -> Final
bigned fs = \(r,next) -> (nth fs (r `mod` (length fs))
       (next r,next))

nth :: [a] -> Int -> a
nth (z:xs) 0 = z
nth (z:xs) n = nth xs (n-1)

prefix :: Int -> Final -> Final
prefix n f = \rng -> (Q n (f rng))
```

The function `(test x m)` defined below calls `m` times our semantic interpreter to execute the program `x`. Each time, the random number generator is initialized with a new (pseudo-)random value. As a consequence, different results (execution traces) are produced at consecutive executions of the same program, thus simulating the non-deterministic behavior of a "real" concurrent system.

```
test :: X -> Int -> IO ()
test x m = aux x m rng0
     where aux x 0 rng = return ()
           aux x m (r,next) =
               do { putStr (show (sem x e0 c0 s0 (r,next))
                ++ "\n\n");
                    aux x (m-1) (next r,next);
                  }
```

We test the denotational prototype for L_{obj} on "real life" programs. The first one is a concurrent generator of prime numbers based on the sieve of Erathostenes. It creates a new object for each prime number. Therefore, in this case the performance degrades continuously. The second example program demonstrates remote object destruction and cloning. A counting object is created and left alone to do its job for a little while. Then, two clones of this object are created and there are three counters working in parallel. After some time, the three objects are killed and the program terminates.

```
x4 = LetRec "drive"
        (Ned [(Snd "c" (V "i"),Seq (Assign "i" (Plus (V "i") (Z 2)))
                                   (Call "drive"))])
     (LetRec "run"
        (Seq (Seq (Ned [(Rcv "i",Skip)])
                  (If (Eq (Z 0) (Mod (V "i") (V "p")))
                      Skip
                      (Ned [(Snd "cout" (V "i"),Skip)])))
             (Call "run"))
     (LetRec "sieve"
      (Seq (Ned [(Rcv "p",Skip)])
      (Seq (Write (V "p"))
      (Seq (New "cout" (Call "sieve")) (Call "run"))))
      (Seq (Assign "i" (Z 3)) (Seq (New "c" (Call "sieve"))
           (Call "drive"))))
x5 = (LetRec "y"
         (Seq (Write (V "v"))
              (Seq (Assign "v" (Plus (V "v") (Z 1))) (Call "y")))
     (LetRec "sleep"
         (If (Lt (Z 0) (V "v1"))
             (Seq (Assign "v1" (Minus (V "v1") (Z 1))) (Call "sleep"))
             Skip)
         (Seq (New "w" (Seq (Assign "v" (Z 10)) (Call "y")))
              (Seq (Seq (Seq (Assign "v1" (Z 3)) (Call "sleep"))
                        (Seq (Clone "w1" "w") (Clone "w2" "w")))
                   (Seq (Seq (Assign "v1" (Z 7)) (Call "sleep"))
                        (Seq (Kill "w") (Seq (Kill "w1")
                                             (Kill "w2")))))))))
```

Now one can perform the following experiments:

```
Main> test x4 1
 3 5 7 11 13  17 19 23 29 31 37 41 43 47 53 59 61 67 71 73{Interrupted!}
```

```
Main> test x5 4
 10 11 12 13 14 13 14 15 15 14 15 16 16 17 16 17 18 18 17 18 19 19

 10 11 12 13 13 13 14 14 15 14 15 15 16 16 16 17 17 18 18 19

 10 11 12 13 13 13 14 14 15 14 15 15 16 16 16 17

 10 11 12 13 13 14 14 14 15 15 15 16 16 16 17 17 17 18 18 18 19 19
```

5. Conclusions and future research

The CSC technique provides a discipline for compositional development of concurrent programming languages based on the concept of a *continuation*. It provides the ability to encapsulate the concurrent behavior in continuations. The semantic model for remote object control given in this paper shows that, by using the CSC technique parallel computations can be manipulated as data in a strict denotational framework. Classic compositional technique do not seem to provide an adequate framework for handling such operations.

In the near future, fundamental research related to the CSC technique will be conducted in two main directions. First, in order to provide an abstract framework for handling context changes and locality in concurrent languages development, we plan to study the possibility of using the CSC technique in the possible world semantics, eventually by extending models given in [12, 2]. Second, in order to improve the flexibility, elegance and modularity of the denotational semantic descriptions, we also plan to study the possibility to define monads [8, 16] for the CSC technique.

References

[1] J.W. de Bakker and E.P. de Vink. *Control flow semantics.* MIT Press, 1996.

[2] S. Brookes. The essence of parallel Algol. *Information and Computation*, vol. 179(1), pages 118–149, 2002.

[3] L. Cardelli. A language with distributed scope. In *Proc. of the 22nd Annual ACM Symposium on Principles of Programming Languages*, pages 286–297, 1995.

[4] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[5] A.A. Holzbacher. A software environment for concurrent coordinated programming. In *Proc. of 1st Int. Conf. on Coordination Languages and Models*, pages 249–267, Springer, 1996.

[6] IBM Aglets website: http://www.trl.ibm.com/aglets.

[7] D. Lauge and M. Oshima. *Programming and deploying Java mobile agents with Aglets.* Addison Wesley, 1998.

[8] E. Moggi. An abstract view of programming languages. Technical Report ECS-LFCS-90-113, University of Edinburgh, 1990.

[9] S. Peyton Jones and J. Hughes (editors). *Report on the programming language Haskell 98: a non-strict purely functional language*, 1999. Available from `http://www.haskell.org/`.

[10] G.D. Plotkin. A powerdomain construction. *SIAM Journal of Computing*, vol. 5, pages 522–587, 1976.

[11] C. Stratchey and C.P. Wadsworth. Continuations: a mathematical semantics for handling full jumps. Technical monograph PRG-11, Programming Research Group, Univ. Oxford, 1974.

[12] R.D. Tennent and J.K. Tobin. Continuations in possible world semantics. *Theoretical Computer Science*, vol. 85(2), pages 283–303, 1991.

[13] E. Todoran. Metric semantics for synchronous and asynchronous communication: a continuation-based approach. In *Proc. of FCT'99 Workshop on Distributed Systems*, *Electronic Notes in Theoretical Computer Science (ENTCS)*, vol. 28, pages 119–146, Elsevier, 2000.

[14] E. Todoran and N. Papaspyrou. Continuations for parallel logic programming, In *Proc. of 2nd International ACM-SIGPLAN Conference on Principles and practice of Declarative Programming (PPDP'00)*, pages 257–267, ACM Press, 2000.

[15] E. Todoran and N. Papaspyrou. Denotational prototype semantics for a simple concurrent language with synchronous communication. Technical report CDS-SW-TR-1-04, National Technical University of Athens, School of Electrical and Computer Engineering, Software Engineering Laboratory, 2004.

[16] P. Wadler. Monads for functional programming. In *Advanced Functional Programming*, Springer, LNCS 925, 1995.

[1] Technical University of Cluj-Napoca, Faculty of Automation and Computer Science, Department of Computer Science, Baritiu Str. 28, Cluj-Napoca, ROMANIA

E-mail address: {`Eneia.Todoran,Cornelia.Melenti`}`@cs.utcluj.ro`

[2] "Babes-Bolyai" University of Cluj-Napoca, Faculty of Mathematics and Computer Science, Department of Computer Science. M. Kogalniceanu Str. 1, Cluj-Napoca, ROMANIA

E-mail address: `florin@cs.ubbcluj.ro`

[3] National Technical University of Athens, Department of Electrical and Computer Engineering, Software Engineering Laboratory, Polytechnioupoli, 15780 Zografou, Athens, GREECE

E-mail address: `nickie@softlab.ntua.gr`

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume L, Number 1, 2005

MODELING CROWD BEHAVIOR USING EMOTIONAL ANTS

SOUMYA BANARJEE, CRINA GROSAN, AND AJITH ABRAHAM

Abstract. It is known that one of the most disastrous forms of collective human behavior is the kind of crowd stampede induced by panic. This situation often leads to fatalities as people are crushed or trampled. This problem has been well researched from a socio-psychological point of view. In this paper we attempt to study and analyze the crowd behavior by using an Ant Colony Optimization (ACO) based computational framework. The initial simulations refer to a panic situation generated in few connected cities of a war affected country.

1. Introduction

When people are part of a crowd, they often behave differently than if they were by themselves. Crowds often act overly frantic or fearful. The shout of 'FIRE!' in a packed movie theater will result in a stampede to the door. Sometimes, crowds behave in a cruel and violent manner. The lynch mobs in America at the turn of the century are a classic example of how barbaric people can be when in a crowd. Crowds can also be apathetic. Often people will act as bystanders to an assault on another person without doing anything to help the person being assaulted.

What is the cause of crowd behavior? This question has intrigued psychologists for decades. The answer appears to depend on the type of crowd behavior and sometimes on the context in which the behavior is shown. Some psychologists suggest that group hostility can arise as a result of de-individualization. De-individualization refers to a weakened sense of personal responsibility. In other words, since people feel anonymous in a crowd, they often act irresponsibly and without care for others.

So, from a modeling perspective study of crowd events is appealing to social simulation researches, because their associated phenomena are largely emergent in nature. The recent lavish development of applied cognitive science [1] helps the researcher to evaluate the other hybrid models related to crowd behavior.

Received by the editors: May 10, 2005.

2000 *Mathematics Subject Classification.* 68T20, 68-02, 68Txx.

1998 *CR Categories and Descriptors.* I.6 [**Simulation and Modeling**]: Simulation Theory - *Model classification*, *System Theory* .

The purpose of the present research is to create an efficient biologically inspired agent (ant colony model) to analyze emotion model of crowd. We expect the proposed model would have a phenomenal impact on many policies and strategies, as the model considers the crowd model and behavior for a war-infected region.

In Section 2, we investigate few existing models of crowd behavior simulation in different aspects of social boundary and their scientific significance which inspired the present work. Section 3 describes the role of bio-inspired agent for the proposed model and, subsequently how the newly introduced 'emotional ants' are constructed to model the crowd behavior during a war situation. Some simulation and illustrations are also provided. Results are analyzed in Section 4 followed by few conclusions in the last Section of the paper.

2. Related Research

There are several significant breakthroughs in the applied cognitive sciences concentrating mainly on the behavioral modeling of crowd using different paradigms. The behavior of pedestrians in different situations has been exhaustively detailed in the literature [2, 3, 4].

Agent based models also have been proposed which begin to address the individual movement. Again a frequently cited model was by Helbing et al. [4]. The model is strongly physics oriented which calculates forces acting on agents to determine movement with excessive forces leading to agent injuries. Similarly there are other works, which describes the crowd modeling simulation in emergency situations [5].

Pedestrian traffic in large cities is modeled in [1]. The work presents a way of simulating an intelligent crowd behavior in a virtual city. There are also some work related to urban planning with the help of crowd movement.

In tune of present research there is substantial motivation to study the psychological part of crowd model. Considering all the major works it has been observed that there is significant gap between the crowd model and their emotion especially in an uncertain situation. The work we propose in this paper refers to a country or region in the war-affected situation and exhibits plenty of cognitive, behavioral and psychological impacts under different conditions.

Crowd event may accumulate and occur in several situations. Some of them can be:

- Panic situation after air raid
- Negotiation between town leaders and soldiers
- Searching for explosives and weapons
- Protest over the arrest essential commodity, if any.
- Domestic terrorist attack or gorilla war resulting looting, shouting slogans heckling etc.
- Even crowd gather after he groups of troops arrived in between.

Therefore, in the situation considered here, the country 'C' comprising different towns, like t_1, t_2,. . . ,t_n are affected with those crowd movements.

3. Proposed Model Using Emotional Ants

The main objective of the proposed model is to fabricate an emotion based analytical model of crowd behavior which in turn seems to be more realistic in uncertain environment. The agent used in this model is biologically inspired whose transition behavior is modeled using fuzzy logic.

In any corner of a region, during war the people moves from one city to another, the city dwellers as crowd also alter their positions. The problem could be more complicated if the third city is also war affected. So, different cognitive behavior of crowd can be modeled using the same model.

- Anger (A)
- Selfish minded (S1)
- Confused (C)
- Sad (S2)

When a selfish minded crowd (only bother about their own shelter) meets an angry crowd from other city, both become confused, because each of their emotion is opposite. We have then:

A + S2 => 2C
S1 + C => 2S1 dominant model
A + C => 2A

Therefore likely behavior of crowd under different situations are as follows:

- The meeting of a selfish minded crowd and a sad minded crowd could result in a confused crowd
- Sad crowd could get angry if they are upset that some crowd components can be selfish when they are sad

In this case we have:

Selfish + Sad => C+A
Selfish + Angry => 2C
Sad + Confused => A +C

Let i and j be subjects of emotions, emotion carriers, person and modules. The subjects take their states s_i and s_j from the set e = {selfish, anger confused, sad} in the discrete time: for $t \in N$, s_i^t, $s_j^t \in E$. The subjects move in a physical space and collide with each other.

When confronted they update their emotional states:

$$S_i^{t+\delta} = f(S_i^t + S_j^t)$$

A relation between the molecules of x and y is seen as:

$$\{(f\ (x,y),\ f(y,x)\} \rightarrow \{z_1,\ z_2\ \},$$

where $x,\ y,\ z_1,\ z_2 \in E$.

On this basis, the war crowd is modeled using an emotional ant agent. We interpret the model of emotion through the extended model of ACO [6] with some flavor of parallel ant agent in a multi agent scenario. The model is well backed up using fuzzy if-then rule templates to monitor the emotional ant movement. This makes the hybrid model more smart to tackle any kind of input conditions.

The pheromone level belonging to one colony has different meaning for other colonies representing different crowd behavior. Therefore the pheromone communication is based on fuzzy if-then rules, by which emotions of ant agents are exchanged.

3.1. **Pheromone Model of Emotional ants.** As the model uses the foraging behavior of ants, the foraging tasks may be abstractly viewed as a sequence of two alternating tasks for each ant: start from a nest where panic situation occurs and moves to a safe place (food). In this problem, we simulate the situation through ant, so the affected city is the start place and the safe place is the goal state or vice versa. The ant agent receives a reward at goal state; at other state does not.

Ants exhibit satisfaction or reward $P(s)$ for transitioning to the desired goal (safe place) irrespective the emotional state of the crowd (selfish or sad and even confused). The utility value of the state Vp (C_i) is the concentration of a given pheromone type p at the city location C_i. Now the difference of pheromone encoding occurs in the following type p at the city location C_i. Now the difference in the pheromone encoding occurs as follows:

- The ant's, strategy as we define
- $(S_i \rightarrow A)$ which maps states into action

The particular choice of pheromone to update and to base transition decision is dependent on the ant's internal state.

(1) We choose the ant agent, released to search the safe place for the crowd problem with *attractive* pheromone
(2) Repulsive pheromone

These ants often select the action putting the pheromone when reaching the 'safe place' and tends to relocate by preferring little intensity of pheromone. If they discover a safe place they keep forcing it and put no pheromone. Through this local behavior, the pheromone space is formed in such a way that the gradient of pheromone density is full towards the safe place.

When it reaches the goal, it increases the value on the track it takes and subsequently this safety condition is exchanged, while visiting other crowd; similarly they also adopt certain route and so on. The main steps are presented below:

for each crowd population **do**
  if safe place found
  then increases edge weights on path to safe place
  else if dead end found
    then
      Population stack until a new route towards safe place is found.
      Decrease weight of edge corresponds to popped node
    else
      Select a neighboring node of the current node of movement of crowd
      Push this node into the stack
    endif
  endif
endfor

The modification in the development of ant system has primarily related to modeling the methods of communication among ant agents. Although substantial progresses have been achieved with the crowd algorithm, but transition rule of next iteration for the ants have remained practically unaltered. Ant colony related algorithm show good performance in solving problems that are combinatorial in nature. However some of the real life problems characterized by uncertainty are not covered by any of the modifications of any system that are found in research literature.

In this work, the transition rules for the ant agent incorporated to investigate the crowd movement are modeled using fuzzy if-then rules. Basically the emotion template designated for selfish, angry, sad or confused crowd are presented through fuzzy rule of following type.

If selfish minded crowd is small and sad crowd is less and trail intensity is stronger then volume of confused crowd becomes very small

Therefore to implement the fury transition rule for the flow of the ant agent in war affected region we use following mathematical back up:

1. The concentration of the trail phenomena c_{ij} on branch i (i=1,2) immediately behind each choice point j (j=1,2) changes in time t according to:

$Dc_{ij'}$ $D_t = q\ \emptyset_{ij}(t) + q\emptyset_{ij}(t\text{-}\tau)$ - $Vc_{ij}(\tau)$

with j' = 3 - j

where:

$\emptyset_{ij}$ (t) represents the overall flow of foragers from the nest to the food source choosing branch i behind the choice point 1, $\emptyset_{12}(t)$ the opposite flow on branch i behind the other choice point j_1= 3 – j= 2, t the average time required for an ant to get from one choice point to the other, q the quality of phenomena laid on

the trail and V the decay rate of the phenomena. Moreover if the density is low enough,

$\emptyset_{ij}(t) = \emptyset_j(t)F_{ij}(t)$;

Where $\emptyset_i$is the outbound flow of foragers from the nest to food source and $\emptyset_2$the opposite nest bound flow. The function F_{ij} describes the relative attractiveness of the trail on branch i at choice point j.

2.The model the pushing of crowd in a panic situation it is important to coordinate the overall flow of ants arriving at choice point j and choosing branch i and the following formula is used:

$\emptyset_{ij}(t) = \emptyset(t)F_{ij}[1\text{- } Ya\ \emptyset_{ij}(t\text{- } \tau)/w] + \emptyset\ (t)F_{ij}(t)\ Ya\ \emptyset\ _{ij}(t\text{- } \tau)/w$

Here $[\emptyset_j(t)\ Fij(t)]$ represents the flow of ants engaged on branch i $[\emptyset_j(t) \quad F_{ij}(t)]$, diminished by the flow of ants pushed towards the other branch i' by ants arriving from the opposite direction.

$a\emptyset_{ij}(t\text{-}\tau)/w$ is the proportion of ants decelerated by interaction.

The factor a is proportional to the interaction time period and the lateral width of ants.

$Y \approx 0.57$ denotes of being pushed in case of crowd encounter.

The 2^{nd} term on the right hand side of the above equation represents the flow of ants that were engaged on branch i and were pushed towards branch i.

3.2. **Details of fuzzy rule based propositions.** In order to incorporate the fuzzy rule [7] (shown in the model) for crowd transition movement through ant agents we assume certain parameters:

d_{ij}^k– the expected distance that k^{th}ant will travel if it decides to go from node i to j .

τ_{ij}–The pheromone trail intensity that the k^{th} ant can smell when traveling between node i and node j.

w_{ij}^k– weight / importance of the k^{th} ant located in node i to visit node j.

Rule 1:

if d_{ij}^k is *small* and τ_{ij} is *weak*

then ants importance/ weight $w_{ij}^k$of visiting j^{th} node is *very high*

else

Rule 2:

if d_{ij}^k is *small* and τ_{ij} is *medium*

then ants importance/ weight d_{ij}^k of visiting j^{th} node is *very high*

else

Rule 3:

if d_{ij}^k is *small* and τ_{ij} is *strong* then ants importance/ weight d_{ij}^k of visiting j^{th} node is *very very high*

else

Rule 4:

if d_{ij}^k is *medium* and τ_{ij} is *weak*
then ants importance/ weight d_{ij}^k of visiting j^{th} node is *low*
else

Rule 5:
if $d_{ij}^k$is *medium* and τ_{ij} is *medium*
then ants importance/ weight d_{ij}^k of visiting j^{th} node is *medium*
else

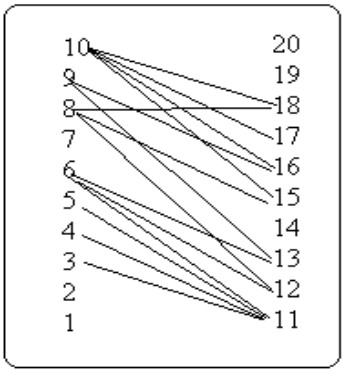
Rule 6:
if d_{ij}^k is *medium* and τ_{ij} is *strong*
then ants importance/ weight d_{ij}^k of visiting j^{th} node is *high*

3.3. **Graphical representation.** The network is composed of few cities and their intra connections. The total number of layers in the network, n is equal to the number of transit lines. There is a strong connectivity between two cities where from the movement of crowd begins.

Let the origin of panic crowd 'o' origin and it has to move to safe city, the ants have few options when choosing the first node in the first layer. We model the emotion of ants to be represented through the deposition of new pheromone and at the same time it reflects the propagation from neighboring places.

The attractive pheromone represents the safe state of crowd. As the crowd may alternate between cities, this leads the system into a bi-partite graph. There are 2 shortest paths through this graph from state 10 (C1, 3, 3) to state 11(C2, 0, 0). So each 11 steps long and differing only the penultimate state (which in this case is safe state of crowd).

FIGURE 1. Graphical representation of the experiment



In our model we follow two different modes:

(1) End search safe place mode

(2) Start search mode

End search mode simply denotes terminate the run at the point concerning the failure. When starting the search node the ant agent test the trip before they leave any city. If the transmission is not achieving safe place then they cancel the pheromone and for next turn to come. We place the ants with different emotions representing pheromone where the transition rules were controlled using fuzzy-if-then rules.

So as base line:

- We consider the entire crowd movement without pheromone (0 bits of pheromone)
- The choice or liking functions return a constant probability
- Afterward agents tries to sense the category of pheromone and category of crowd emotion e.g. Selfish, angry or confused or sad crowd
- From the emotion template of the crowd model presented, we can declare selfish and angry crowd lay down identical (as their output state remains always confused). So an ant agent can know the selfish and angry crowd is present from the pheromone deposition, not the density or population estimation

To sum up these parameters we solicit the model where the dynamics of emotion propagates from one place to the neighboring places. Pheromone will be deposited (irrespective of the current state of crowd that will be clarified further on the basis of their emotion) at every node that was visited at least by one ant agent. Therefore at the beginning of the search process it will be assumed that the pheromone trail is very low in every node and it is equal to some positive constant. But it is not certain what should be the density of crowd in each city at any point of time. In 2 the flow of experiment is presented.

Results of experiment are presented in Table 1.

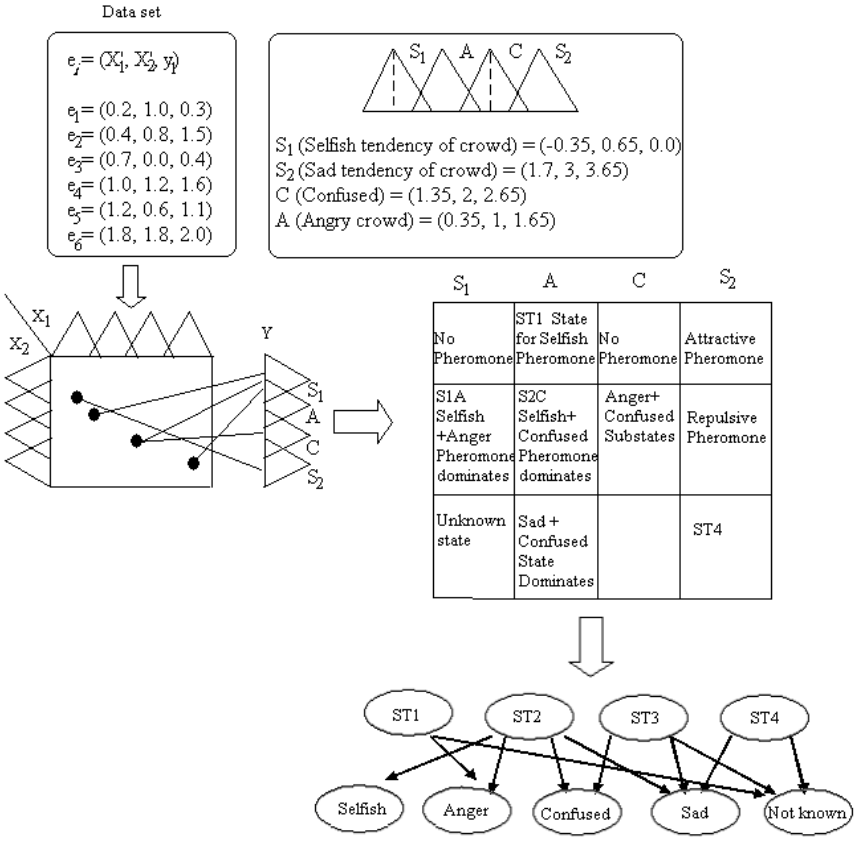
3.4. **Synthetic Pheromone.** Insects perform impressive feats of coordination without direct inter-agent coordination, by sensing and deposition pheromones in the environment [8]. For example ants construct networks of paths that connect their nests with available food sources.

Mathematically, these networks from spanning trees, minimizing the energy ants expend in bringing food into the nest.

The real world extrapolates three operations on chemical pheromones that support purposive insect actions:

- It aggregates deposits from individual agents
- Pheromone evaporates over time
- It diffuses pheromones to nearby places, thus disseminating information for access by nearby agents.

FIGURE 2. Different state of pheromone with the metal tendency of crowd



The pheromone field constructed by the ants in the environment is in fact a potential field that guide their movements. The dynamic field is a measure of agent movement. Agents increment the dynamic field level when moving. By analogy with ant pheromones the dynamic field diffuses and evaporates. By consulting the dynamic field an agent can follow other nearby agents without directly considering the position of any other agent.

Considering these diversified applications of synthetic pheromone, there are different domains applications [8]. The potential of insect models for multivalent coordination and control thus is receiving significant attention.

TABLE 1. Experiment results

		End search safe place mode				**Start search safe place mode**	
Emotion position	Bits of information	Success for safe place	Average length	Maximum length	Median	Lower quartile	Upper quartile
Neutral Ants In crowd	**0**	**0**	**0**	**-**	**353**	**198**	**738**
Ants Generate Selfish Pheromone	**1**	**7**	**1**	**20**	**85.5**	**57**	**124**
Ants Generate Anger Pheromone	**1**	**5**	**2**	**21**	**74**	**57**	**112**
Ants Generate Confused Pheromone	**3.58**	**49**	**6**	**21**	**78**	**44**	**128**
Ants Generate Sad Pheromone	**3.58**	**18**	**4**	**20**	**73.5**	**46**	**117**
Optimum Safe place, combined all of them (Smart or attractive Pheromone)	**4.91**	**8940**	**755**	**93**	**23.5**	**18**	**34**

Distributed coordination problem is one of the aspects of those applications. Subsequently there are the issues of optimization techniques under the same problem domain. To tackle the uncertainty of search space and constraints there are several techniques proposed [9][10][11]. Hence, the problem of tackling the distribution model of emotion representation of crowd is equally challenging in the light of synthetic pheromone distribution techniques.

The underlying mathematics of the entire search in the war affected region would be:

$S(t + 1, p) = E * s(t, p) + r\ (t, p) + q(t, p)$

Where,

- $P = \{p_i\}$ = set of places
- N: $P \rightarrow P$ = neighbor relation between places.
- $S(t,p)$ = pheromone strength at time t and place p

- $r(t,p)$ = external input at time t to place p
- $q(t,p)$ = propagated input at time t to place p
- E (0, 1) = Evaporation parameter.

4. Discussions

In the work we specified different pheromone values as indicated the state of mind of the crowd. Typically, 4 types of pheromone representations have been adopted along with no pheromone and smart pheromone (when the different behavioral crowd reach safe place in a war affected region).

Using the proposed model with the smart/attractive pheromone and on the basis of fuzzy rule transition, end search safe place mode succeeds 8940 times, i.e. the crowd is most safe with this choice combining their mental state, 755 with the minimum length and maximum length of 93. The start search safe place mode median is 23.5 with quartile of 18 and 34 respectively.

The improvement over the other types of individual pheromone (e.g. sad pheromone) is (73.5 - 23.5)/ 73.5 = 68%. So it implies that even with a poor initial start which does not exhibits good performance, as more information is available the perofmance can enhance significantly due to the modification of the different combination of pheromone values of the mental states of the crowd.

5. Conclusion

An effort is made to simulate the emotional model of crowd using bio-inspired agents and meta-heuristic approaches. It is expected that this kind of model would be able to assist substantially in social science, cognitive science and broadly machine like behavior and learning.

References

[1] Funge, J., Making Them Behave: Cognitive Models for Computer animation, Ph.d Thesis University of Torrento, 1998.
[2] Helbing. D, A Mathematical Model for the Behavior of Pedestrians, 1991.
[3] Helbing. D, Farkas. I.J., Molner, P., Viesek, T: Simulation of pedestrians crowds in normal and evacuation situations in: Schreckenberg, M, Sharma S.D.(eds.): Pedestrian and evacuation dynamics, Springer-Verlag, New York.
[4] Helbing. D, Fakas. I, Viesek T: Simulating Dynamical Features of Escape Panic. Nature v. 407, pp. 487-490, 2000.
[5] Oleg Yacovenko and Phdvalery Startrov, Crowd Behavior Modeling in Emergency.
[6] E . Bonabeau, Dorigo. M and Theraulaz G, Swarm Intelligence: From Natural To Artificial System. New York, Oxford University Press, 1999.
[7] Casillas. J, Cordon. O, Herra. F and Magdalena. L, editors: Interpretability issues in Fuzzy Modeling, Springer-Heidelberg, Germany, 2003.
[8] Brueckner S, Return from the Ant: Synthetic Ecosystems for Manufacturing Control. Thesis at Humboldt University, Berlin Department of Computer Science, 2000.
[9] Parunak, H.V.D , Adaptive control of Distributed Agents through Pheromone Techniques and Interactive Visualization. 2000, web:www.erim.org.cec/projects/adaptiv/

[10] Horst R, Pardalos, P.M. editors, Handbook of Global Optimization, Kluwer Academic Publishers, 1995
[11] Horst R, Tuy, P. Global Optimization: Deterministic Approaches. Springer-Verlag, 1990.
[12] Mockus, J, Baysian Approach to Global Optimization: Theory and Applications. Kluwer Academic, 1989.

Department of Computer Applications, Institute of Management Studies, India
E-mail address: soumyabanerjee@imsddun.com

Department of Computer Science, Babes-Bolyai University, 400084 Cluj-Napoca, Romania
E-mail address: cgrosan@cs.ubbcluj.ro

School of Computer Science and Engineering, Chung-Ang University, Korea
E-mail address: ajith.abraham@ieee.org

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume L, Number 1, 2005

LARGE CANDIDATE BRANCH-BASED METHOD FOR MINING CONCURRENT BRANCH PATTERNS

JING LU, OSEI ADJEI, WEIRU CHEN, FIAZ HUSSAIN, CĂLIN ENĂCHESCU, AND DUMITRU RĂDOIU

Abstract. This paper presents a novel data mining technique, known as Post Sequential Patterns Mining. The technique can be used to discover structural patterns that are composed of sequential patterns, branch patterns or iterative patterns. The concurrent branch pattern is one of the main forms of structural patterns and plays an important role in event-based data modelling. To discover concurrent branch patterns efficiently, a concurrent group is defined and this is used roughly to discover candidate branch patterns. Our technique accomplishes this by using an algorithm to determine concurrent branch patterns given a customer database. The computation of the support for such patterns is also discussed.

Keywords: Post Sequential Patterns Mining; Concurrent Branch Patterns; Sequential Patterns Mining

1. Introduction

Sequential patterns mining proposed by Agrawal and Srikant [1] is an important data mining task and with broad applications. Based on the analysis of sequential patterns mining, we proposed a novel framework for sequential patterns called sequential pattern graph (SPG) as a model to represent relations among sequential patterns [2]. SPG can be used to represent sequential patterns encountered in patterns mining. It is not only a minimal representation of Sequential patterns mining result, but it also represents the interrelation among patterns. It establishes further the foundation for mining structural knowledge. Based on SPG and sequential patterns mining, a new mining technique called post sequential patterns mining (PSPM) [3] is presented to discover new kind of structural patterns. A structural pattern [4] is a new pattern, which is composed of sequential patterns, branch patterns or iterative patterns.

In order to perform post sequential patterns mining, the traditional sequential patterns mining should be firstly completed. Post sequential patterns mining can be viewed as a three-phase operation that consists of *pre-processing*, *processing* and *post-processing* phases. In the pre-processing phase, based on the result of sequential patterns mining, the Sequential Patterns Graph (SPG) is constructed. SPG is a bridge between traditional sequential patterns mining and the novel post

Received by the editors: June 1, 2005.

sequential patterns mining. The processing phase corresponds to the execution of the mining algorithm, given the maximal sequences set (MSS) recognized by SPG and customer sequence database DB as input, structural patterns (including concurrent branch patterns, exclusive branch patterns and iterative patterns) are discovered. During post-processing, the mined structural pattern can be represented graphically. In this paper, we focus on concurrent branch pattern and its mining algorithms. We address the question: *Given a set of sequential patterns and customer sequence database DB, how can we efficiently find concurrent branch patterns?*

The first step in the discovery of a concurrent process should be to identify the individual threads and their individual behaviours. Our work demonstrates that since it is part of the post sequential patterns mining, concurrent branch pattern mining discovers patterns on the basis of sequential patterns. In a concurrent process, it is important to also locate the points where the threads interact. Our method solves this crucial problem by taking out a common prefix and/or a common postfix from sequential patterns that is candidate branch patterns. Section 3, discusses the concurrent branch pattern mining algorithm whilst section 4, reviews some related work. Section 5 presents our conclusions.

2. Problem Statement

To formally define the concurrent branch mining algorithm we introduce some basic terminology. In the following definition, let SP represent Sequential Patterns; $x\alpha$, $x\beta$, αy, βy, $x\alpha y$, $x\beta y \in SP$; α, $\beta \in SP$; $x, y \in SP$ or $x, y \in \emptyset$.

Definition 1: CandidateBranch Pattern

Sequential patterns which contain common prefix and/or common postfix can constitute *Candidate Branch Pattern.*

- Sequential patterns $x\alpha$ and $x\beta$ can make up a candidate branch pattern which has a sub-sequence x as a common prefix and denoted by $x[\alpha, \beta]$.
- Sequential patterns αy and βy can make up a candidate branch pattern which has a sub-sequence y as a common postfix and denoted by $[\alpha, \beta]y$.
- Sequential patterns $x\alpha y$ and $x\beta y$ can make up a candidate branch pattern which has sub-sequence x as a common prefix, sub-sequence y as a common postfix and denoted by $x[\alpha, \beta]y$.

In the above definitions, notation $[\alpha, \beta]$ represents two *branches* of a candidate branch pattern.

Let us consider some examples. Sequential patterns <*efcb*> and <*ebc*> can make up a candidate branch pattern which has e as a prefix and denoted by $e[fcb,bc]$. This candidate branch pattern has two branches, *fcb* and *bc*. Sequential patterns <*fcb*>, <*dcb*> and <*acb*> can make up a candidate branch pattern which has *cb* as a postfix and denoted by [*f*, *d*, *a*]*cb*. This candidate branch pattern has three branches *f*, *d* and *a*. It should be noted that in a candidate branch pattern such as $a[b,c]d$, the order of b and c is indefinite. Therefore, $a[b,c]d$ can appear in a transaction database in the form of *abcd*, *acbd* or *a(b,c)d*. The purpose of defining

a candidate branch pattern is to discover true branch patterns (concurrent branch patterns or exclusive branch patterns). A candidate branch pattern can also be extended to multiple sequential patterns.

Definition 2: Concurrence

The *concurrence* of sub-sequential patterns α and β is defined as the fraction of customers that contain α and β simultaneously and it is denoted as:

$$concurrence(\alpha \wedge \beta) = \|\{T : \alpha \cup \beta \subseteq T, T \in D\}\|/\|D\|$$

Let *minsup* be user specified minimal support, if *concurrence* $(\alpha \wedge \beta) \geq$ *minsup* is satisfied then α and β are *concurrent*. Similarly, multiple candidate branches $a_1 \ldots a_i$ ($\alpha_i \in SP$; $1 \leq i \leq n$) are concurrent branches if and only if *concurrence* $(a_1 \wedge \ldots \wedge a_i) \geq$ *minsup*.

Definition 3: Concurrent Branch

Two branches α and β of candidate branch pattern $x[\alpha, \beta]y$ are *concurrent branch* if and only if in a transaction database, α and β are concurrent between a common prefix x and/or common postfix y.

Definition 4: Concurrent Branch Pattern

For candidate branch pattern $x[\alpha, \beta]y$, if branches α and β are concurrent branches, then $x[\alpha, \beta]y$ is *concurrent branch pattern*.

The **problem of concurrent branch pattern mining** is to *find the complete set of concurrent branch patterns in a given sequential pattern mining result and customer sequence database DB with respect to given support threshold.*

Example 1: Let us consider a customer sequence database in PrefixSpan [5]:

(1) <a (a,b,c) (a,c) d (c,f)>
(2) <(a,d) c (b,c) (a,c)>
(3) <(e,f) (a,b) (d,f) c b>
(4) <e g (a,f) c b c>

and two branches *f* and *eb* of the candidate branch pattern [*f*, *eb*]*c*. Let *minsup*=50%.

Both customer sequence (3) <(e,f) (a,b) (d,f) c b> and (4) <e g (a,f) c b c> contain *f* and *eb*. Thus, the concurrence ($f \wedge eb$) is 50%. That is, *f* and *eb* are concurrent branches and *sup* ([f,eb]c)=50%. Therefore, the candidate branch pattern [*f*, *eb*]c is a concurrent branch pattern.

It can be concluded from definition 4 that, concurrent branch patterns mining problem can be decomposed into the following sub-problems of: how to generate all *candidate branch pattern*; how to determine the *concurrence* of candidate branches; and how to calculate the *support* of candidate branch pattern.

3. Concurrent Branch Pattern Mining

Let us consider the first problem in concurrent branch patterns mining, i.e., how to generate all candidate branch patterns. Since the concurrent branch pattern mining is based on the result of sequential patterns mining, which is the set of sequential pattern, hence the direct way to discover candidate branch pattern should be based on sequential pattern set. All candidate branch patterns can be

generated by taking out a common prefix or/and a common postfix from sequential pattern set. However, the shortcoming of this method is that some non-concurrent branches can be generated.

In order to get rid of non-concurrent items, the concurrent group and the maximal concurrent group are defined first. Then, rough concurrent branch patterns are computed based on the maximal concurrent group to obtain candidate branch patterns.

3.1. Concurrent Group and Rough Concurrent Branch Pattern.

Definition 5: Concurrent Group (*CG*)

Given customer sequences database *DB*, set of items (or itemset) that have transaction support above *minsup* makes up a *concurrent group* and it is denoted by *CG* for brief.

Definition 6: Maximal Concurrent Group (*MCG*)

A concurrent group is called a *maximal concurrent group* if any of its superset is not a concurrent group. The set of maximal concurrent group set is denoted by ***MCGS*** for abbreviation.

Example 2: Consider the customer sequences in example 1 and let *minsup* be 50%. Items (or itemset) sets {a,b,c,d}, {(a,b),c,d,f} and {(a,c),b,d} are all examples of concurrent group since the condition in definition 5 is satisfied. From definition 5 we know that concurrent group is a set and the elements in this set can be an item or an itemset. Consider {(a,b),c,d,f} for example, four elements are contained in this concurrent group, one is an itemset (a,b) and the other three are items c,d, and f. Among these three concurrent groups, {(a,b),c,d,f} is a maximal concurrent group but {a,b,c,d} is not, since its superset {(a,b),c,d,f} is a concurrent group.

If each customer sequence is considered as a transaction, then discovering concurrent group from customer sequence database is identical to the discovery of frequent patterns. The maximal concurrent group of the above example is:

$$MCG = \{\{(a,b),c,d,f\},\{(a,c),(b,c),d\},a,b,c,e,f\}\}$$

Following the definition of the maximal concurrent group, we investigate the relation between the maximal sequence set (MSS) discovered in sequential patterns mining and the maximal concurrent group proposed.

Definition 7: Rough Concurrent Branch Pattern (*RCBP*)

Let C be a maximal concurrent group in *MCG*. *Concurrent sequences* can be obtained by the *sequential intersection* operation of C and each element in *MSS* respectively. These concurrent sequences constitute a rough concurrent branch pattern and denoted by *RCBP* for brief. Sequential intersection operation can be treated as a normal intersection, and the sequence relations among elements after this operation will be consistent with that in the original sequence pattern. The notation for sequential intersection is:

Sequential pattern or Sequential pattern set $\cap$ *Concurrent Group*

Rough Concurrent Branch Pattern is a candidate branch pattern, which has a null common prefix and a null common postfix.

Algorithm 1: Cal_RCBP (Getting a RCBP)
Input: Maximal concurrent group C and maximal sequence set *MSS*.
Output: Rough Concurrent Branch Patterns *RCBP(C)*.
Method: Find the rough concurrent branch patterns in the following steps:

(1) Let rough concurrent branch pattern for *C, RCBP(C)*, be empty.
(2) For each element *ms* in *MSS*
    Add *ms* to *RCBP(C)*;
    For each element (item or itemset) i in *ms*, test if i is an element of C or i is included in one element of C;
    If neither condition is satisfied, then delete i from *ms*.
(3) Delete the element in *RCBP(C)* which contained by another pattern in the *RCBP(C)*.
(4) The result is *RCBP(C)*.

Example 3: Given *MSS*={<eacb>, <efcb>, <a(b,c)a>, <(a,b)dc>, <fbc>, <(a,b)f>, <ebc>, <dcb>, <abc>, <acc>, <(a,c)>} and maximal concurrent group *MCG* = {{(a,b), c, d, f}, {(a,c), (b,c), d}, {a, b, c, e, f}}.

MCG	RCBP
{(a,b),c,d,f}	{<acb>, <fcb>,<aca>,<(a,b)dc>,<fbc>,<(a,b)f>, <dcb>,<abc>,
{(a,c),(b,c),d}	{<acb>, <a(b,c)a>,<aba>,<aca>,<adc>,<bdc>, <dcb>, <abc>,
{a,b,c,e,f}	{<eacb>,<efcb>,<aba>,<aca>,<fbc>,<af>,<bf>,<ebc>,<abc>,

TABLE 1. Rough Concurrent Branch Pattern Example

The rough concurrent branch patterns can be computed using algorithm 1. The final result is shown in table 1.

3.2. **Sub-customer sequence set.** The feature of our method is that the customer sequence database *DB* is not used for counting support after the discovering of candidate branch patterns. Rather, the sub customer sequence set *SubDB* is used for this purpose. The number of entries in *SubDB* may be smaller than the number of transaction in *DB*. In addition, each entry may be smaller than the corresponding transaction because the items (itemset) before the prefix element or after the postfix element are deleted.

Definition 8: Sub-customer sequence set

Given a candidate branch pattern $x[\alpha, \beta]y$ and a customer sequence database *DB*, the *sub customer sequence set* of *DB* is obtained by deleting the *minimal pre-sub sequence* contains prefix x or/and the *minimal post-sub sequence* contains a postfix y of each customer sequence in *DB*. This is denoted by *SubDB*(x,y).

The support of the sub-customer sequence set *SubDB*(x,y) is:

$$sup(\text{SubDB}(x,y)) = |\text{SubDB}(x,y)|/|\text{DB}|$$

Explanation. Minimal pre-sub sequence contains x: Suppose x=*cd* and the customer sequence is *acbdefdg*. The result for deleting the minimal pre-sub sequence is *efdg*.

Minimal post-sub sequence contains y: Suppose y=*bg* and customer sequence is *acbdefdg*. The result for deleting the minimal post-sub sequence is *ac*.

The purpose of finding the sub-customer sequence set is to calculate the *support* of the candidate branch pattern $x[\alpha, \beta]y$ and to determine the *concurrence* of branches $[\alpha, \beta]$. For a candidate branch pattern $x[\alpha, \beta]y$, (i) if *sup*(SubDB(x,y))< *minsup*, then the candidate branch pattern $x[\alpha, \beta]y$ cannot be a concurrent branch pattern; (ii) if *sup*(SubDB(x,y))≥ *minsup*, then only the *concurrence* checking of branches x and y is needed. That is, it is only necessary to check if x and y occurs simultaneously in each sub customer sequence of *SubDB*(x,y).

Algorithm 2: Gen_SubDB (Computes Sub-customer sequence set)
Input Common prefix x and/or common postfix y of candidate branch pattern $x[\alpha, \beta]y$; Customer sequence database *DB*.
Output Sub customer sequence set *SubDB*.
Method:

SubDB(x,y)=∅;
For each customer sequence *cs*∈*DB* **Do**
{ Scan *cs* from left to right, find the sub customer sequence which contains prefix x completely, record the position p of the last matched element in *cs*. If not found, set p be the length of *cs*;
Scan *cs* from right to left, find the sub customer sequence which contains prefix y completely, record the position q of the first matched element in *cs*. If not found, set p be 0;
If p≥q //There is no sub sequence have prefix x and postfix y in *cs*
then *DB*=*DB*-{*cs*} //Delete *cs* from *DB*
else
Delete sub customer sequence before the p^{th} (contains p^{th}) element and after the q^{th} (contains q^{th}), obtained *cs*(*xy*)
SubDB(x,y)= *SubDB(x,y)*∪ *cs*(*xy*)
End if
}
return *SubDB(x,y)*.

Theorem 1: Given a candidate branch pattern $x[\alpha, \beta]y$ and sub customer sequence set *SubDB(x,y)*, if α and β are *concurrent* in *SubDB(x,y)* i.e., if the number of occurrence of α and β simultaneously in *SubDB(x,y)* is greater than or equal to a user specified minimal support *minsup*, then the candidate branch pattern $x[\alpha, \beta]y$ is a concurrent branch pattern. (Proof is omitted for brevity)

Thus, the problems of how to determine the concurrence of candidate branch pattern and how to calculate the support of a candidate branch pattern reduces to

how to find a sub-customer sequence set *SubDB* and how to check the concurrence of a candidate branch pattern in *SubDB*.

3.3. **Concurrent Branch Pattern Mining Method.** Steps taken to mine concurrent branch patterns based on candidate branch pattern are given as follows.

(1) Find the maximal sequence set (*MSS*) from customer sequences using traditional sequential patterns mining algorithm;

(2) Find the maximal concurrent group set (*MCGS*) from customer sequences in *DB* using traditional frequent patterns mining algorithm;
(3) Generate the rough concurrent branch pattern (*RCBP*) using *Cal_RCBP* algorithm;
(4) Calculate the sub-customer sequence set (*SubDB*) using *Gen_SubDB* algorithm;
(5) Determine the support of the candidate branch in the sub-customer sequence set to generate concurrent branch pattern.

Example 4: Given a customer sequence database *DB* (refer to example 1) and its rough concurrent branch pattern *RCBP* (shown in table 1), steps taken to find all concurrent branch patterns are as follows:

(1) Generate the candidate branch pattern *CanBP* based on *RCBP*. With respect to Table 1:
RCBP(3) = {<eacb>, <efcb>, <aba>, <aca>, <fbc>, <af>, <bf>, <ebc>, <abc>, <acc>}.
Since <*eacb*> and <*efcb*> have a common prefix *e* and a common postfix *cb*, these two sequential patterns can constitute candidate branch pattern *e*[*a*,*f*]*cb*; <*aba*> and <*aca*> have a common prefix *a* and common postfix *a*, and make up a candidate branch pattern *a*[*b*,*c*]*a*; Similarly, <*af*> and <*bf*> make up [*a*,*b*]*f*.
The candidate branch pattern set of RCBP(3) is CanBP_3 = {e[a,f]cb; a[b,c]a; a[b,c]c; ab[a,c]; ac[a,c]; [f,e,a]bc}. In the same way, CanBP_1 = {ac[a,b,c]; [a,f,d]cb; (a,b)[dc,f]; a[b,c]c; [a,f]bc; f[cb,bc]}; CanBP_2 = {ac[a,b,c]; ab[a,c]; [a,d]cb; a[b,c]a; a[b,d,c]c; [a,b]dc}
(2) Generate the Sub-Customer Sequence Set *SubDB*
For the common prefix and/or common postfix in the above candidate branch patterns, the sub-customer sequence set of example 1 can be calculated by using algorithm 2. The result is shown in table 2.
(3) Counting the support to find Concurrent Branch Pattern *CBP*
Calculate the support of the candidate branch in sub-customer sequence set *SubDB* to generate concurrent branch patterns. Here, we only consider: CanBP_1={ac[a,b,c];[a,f,d]cb;(a,b)[dc,f];a[b,c]c;[a,f]bc;a[b,c]a }. Table 3 is an example of the processes involved in the calculation of the support.

Next, the candidate branch which is not concurrent and the number of which is at least 2 is decomposed. The concurrence of its decomposition is determined

Prefix	Postfix	SubDB			
e	cb	(a,b) (d,f)	G (a,f)		
a	c	(a,b,c) (a,c) d	C (b,c)	(d,f)	c b
ab	-	(a,c) d (c,f)	(a,c)	c	
ac	-	(a,c) d (c,f)	(b,c) (a,c)	b	b c
(a,b)	-	(a,c) d (c,f)	(d,f) c b		
-	cb	(a,d)	(e,f) (a,b) (d,f)	e g (a,f)	
-	bc	a	(a,d) c	(e,f)	e g (a,f) c

TABLE 2. Sub Customer Sequence Set of Example 1

Prefix	Branches	Postfix	support in Sub customer sequence set	Concurrence
ac	a,b,c	-	1	No
a	b,c	c	3	Yes
-	f,a	bc	1	No
-	a,f,d	cb	1	No
(a,b)	dc,f	-	2	Yes
a	b,c	a	2	Yes

TABLE 3. Example for counting support for $CanBP_1$

continuously. Since $ac[a,b,c]$ is not concurrent, it is decomposed into [a,b], [a,c], [b,c]. Also [a,f,d]cb is not concurrent and it is decomposed into [a,f], [a,d], [f,d]. The process is shown in table 4.

Finally, the concurrent branch pattern CBP_1 derived from $CanBP_1$ is computed as CBP_1={a[b,c]c, (a,b)[dc,f], ac[a,c], ac[b,c], [a,d]cb, [a,f]cb, a[b,c]a}.

4. Related Work

Concurrency is particularly a difficult aspect of some systems' behaviours. Cook *et al.* [6] presented a technique to discover patterns of concurrent behaviour from traces of system events. The technique uses statistical and probabilistic analyses to determine when a concurrent behaviour occurs, and what dependent relationships might exist among events. The technique is useful in a wide variety of software engineering tasks that includes, re-engineering, user interaction modelling, and software process improvement.

Prefix	Branches	Postfix	support in Sub customer sequence set	Concurrence
ac	a,b	-	1	No
ac	a,c	-	2	Yes
ac	b,c	-	2	Yes
-	a,f	cb	2	Yes
-	a,d	cb	2	Yes
-	f,d	cb	1	No

TABLE 4. Example for counting support for the decomposition of *CanBP*

Other related work can also be found in the area of workflow data analysis, since many workflows exhibit concurrent behaviour. Herbst [7-9] investigated the discovery of both sequential and concurrent workflow models from logged executions. Agrawal *et al.* [10] investigated production activity dependency graphs from event-based workflow logs that had already identified the partial ordering of concurrent activities.

5. Conclusions

In this paper, we developed candidate branches based method to detect concurrent behaviour in customer sequence database and to infer a model that describes the concurrent behaviour. The problem of finding concurrent branch pattern was first introduced in this paper. This problem is concerned with finding concurrent branch pattern in a given sequential pattern mining result and a customer database. The main purpose of Post Sequential Patterns Mining is to discover the hidden structural patterns in event-based data. Concurrent branch pattern is an important pattern, which occurs in many event-based data. Thus, we concentrated on concurrent branch pattern mining in this paper.

An important phase for our work is to perform more experiments to support our theories. In our previous work [2], we implemented the algorithm for constructing SPG and analysed the efficiency of that approach. In our existing research work, we anticipate that more experiments are needed to demonstrate the affective nature and efficiency of concurrent branch patterns mining algorithms. This paper has been theoretical; experimentation is on going to establish the validity of our algorithms. In addition to the above, we intend to extend the method to cover concurrent branch patterns to exclusive branch patterns mining or iterative patterns mining. This, we envisage will be our ultimate goal.

References

[1] Agrawal, R. & Srikant, R. (1995). Mining sequential patterns. *Proceedings. of the Eleventh Internal Conference on Data Engineering*(pp.3-14). Taipei, Taiwan. IEEE Computer Society Press.

[2] Lu,J., Wang, X.F., Adjei, O., & Hussain, F.(2004a) *Sequential Patterns Graph and its Construction Algorithm.* Chinese Journal of Computers. *6*, 782-788.

[3] Lu, J., Adjei, O., Wang, X.F., & Hussain, F. (2004b) Sequential Patterns Modeling and Graph Pattern Mining. *Proceedings of the Tenth International Conference IPMU* (pp.755-761). Perugia: Italy.

[4] Lu,J., Adjei, O., Chen, W.R., & Liu, J. (2004c) Post Sequential Pattern Mining: A new Method for discovering Structural Patterns. *Proceedings of International Conference on Intelligent Information Process* (pp.239-250). Beijing: China.

[5] Pei,J., Han, J.W., Behzad M.A, & Pinto, H.(2001). PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. *Proceedings. of 17th International Conference on Data Engineering* (pp.215-226). Heidelberg: Germany.

[6] Cook ,J.E., & Wolf ,A.L. (1998). Event-Based Detection of Concurrency. *Proceedings of the Sixth International Symposium on the Foundations of Software Engineering* (pp.35-45). Orlando: FL.

[7] Herbst, J.(1999). Inducing Workflow Models from Workflow Instances. *Proceedings of the Sixth European Concurrent Engineering Conference*, *Society for Computer Simulation* (pp.175-182).

[8] Herbst, J.(2000a). A Machine Learning Approach to Workflow Management. *Proceedings of European Conference on Machine Learning*(pp.183-194). Lecture Notes in Artificial Intelligence Nr. 1810.

[9] Herbst, J.(2000b). Dealing with Concurrency in Workflow Induction. *Proceedings of the Seventh European Concurrent Engineering Conference, Society for Computer Simulation* (pp.169- 174).

[10] Agrawal, R., Gunopulos, D., & Leymann, F. (1998). Mining Process Models from Workflow Logs. *Proceedings of the Sixth International Conference on Extending Database Technology* (EDBT).

Department of Computing and Information Systems, University of Luton, Park Sq. Luton, LU1 3JU UK, School of Computer Science & Technology, Shenyang Institute of Chemical Technology, Shenyang 110142 China
E-mail address: jing.lu@luton.ac.uk

Department of Computing and Information Systems, University of Luton, Park Sq. Luton, LU1 3JU UK
E-mail address: osei.adjei@luton.ac.uk

School of Computer Science & Technology, Shenyang Institute of Chemical Technology, Shenyang 110142 China
E-mail address: willc@mail.china.com

Department of Computing and Information Systems, University of Luton, Park Sq. Luton, LU1 3JU UK
E-mail address: fiaz.hussain@luton.ac.uk

School of Computer Science, Petru Maior University, Târgu Mureş, Romania
E-mail address: ecalin@upm.ro

School of Computer Science, Petru Maior University, Târgu Mureş, Romania
E-mail address: dumitru.radoiu@Infopulse.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume L, Number 1, 2005

A NEW DYNAMIC EVOLUTIONARY CLUSTERING TECHNIQUE. APPLICATION IN DESIGNING RBF NEURAL NETWORK TOPOLOGIES. II. NUMERICAL EXPERIMENTS

D. DUMITRESCU AND KÁROLY SIMON

Abstract. Recently a new evolutionary optimization metaheuristics, the Genetic Chromodynamics (GC) has been proposed. Based on this metaheuristics a dynamic clustering algorithm (GCDC) is proposed. This method is used for designing RBF neural network topologies. Complexity of these networks can be reduced by clustering the training data. The GCDC technique is able to solve this problem. In Part I the GCDC technique is presented. It is described, how this method could be used for designing optimal RBF neural network topologies. In Part II some numerical experiments are presented. The proposed algorithm is compared with a static clustering technique, the generalized k-means algorithm.

Keywords and phrases: Dynamic evolutionary clustering, Genetic Chromodynamics, designing neural networks, RBF neural networks.

1. Introduction

Recently a new evolutionary search and optimization metaheuristics - called Genetic Chromodynamics (GC) (see [4, 14]) - has been proposed. Based on this theory a clustering method is proposed. This GC-based dynamic clustering technique - called GCDC - is described in [9]. The proposed algorithm can be successfully used for designing optimal RBF neural network topologies.

In this Part some numerical experiments and obtained results are presented. GCDC is used for clustering two-dimensional input data. The use of GCDC for

Received by the editors: February 18, 2005.

2000 *Mathematics Subject Classification.* 68T05, 68T20, 91C20, 92B20.

1998 *CR Categories and Descriptors.* I.2.6 [**Artificial Intelligence**]: Learning – *Connectionism and neural nets*; I.5.3. [**Pattern Recognition**]: Clustering – *Algorithms* .

designing optimal RBF neural network topologies is investigated. The method is compared with a static clustering technique, the generalized k-means algorithm [17].

In the next section the GCDC method is tested on two-dimensional input data. The behavior of the fitness function is investigated. Section 3 presents how this method can be used for designing RBF neural networks. GCDC is used for clustering training data. The topology of the RBF network is designed based on the obtained results. In the experiment presented in Section 4 the GCDC method is compared with the generalized k-means clustering algorithm .

2. Experiment 1

From the two-dimensional input space 19 data points ((x, y) pairs, where $x \in \{100, ..., 300\}$ and $y \in \{100, ..., 300\}$) organized in 5 clusters are considered.

GCDC is used for clustering this data set. The parameters of the method are:

- initial population size: 38;
- parameters for the fitness function: $\alpha = 2, C = 140$;
- mutation step size: $\sigma = 10$;
- merging radius: $\varepsilon = 25$.

After 45 iterations the correct number of clusters is determined by the GCDC method. The algorithm detects existing clusters and corresponding centers. The obtained results are presented in Figure 1.

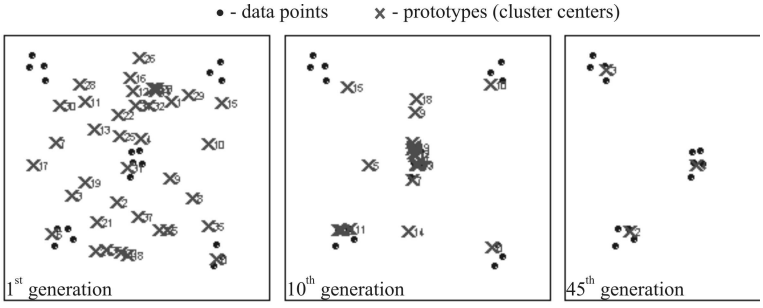


Figure 1. Convergence of the GCDC algorithm: two-dimensional input data, 19 data points organized in 5 clusters

More tests with different parameters for the fitness function are performed. The behavior of the fitness function is presented in Figure 2, Figure 3, Figure 4 and Figure 5.

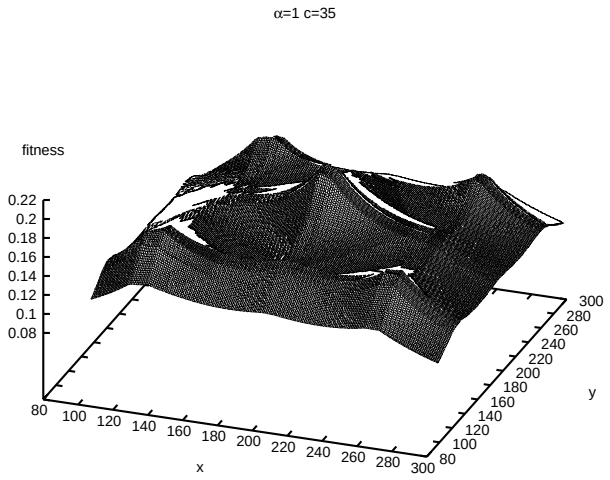


FIGURE 2. Fitness landscape for $\alpha = 1, C = 35$

3. Experiment 2

RBF neural network is used for approximating the function:

$$f : [0, 9.5] \to R, f(x) = 2 \cdot \sin\left(\ln(x) \cdot e^{\cos\left(\frac{x}{2}\right)}\right).$$

3.1. **Experimental Conditions.** From the interval [0,9.5] 200 points are considered as training samples. GCDC is used for clustering training data.

The obtained centers are used as center parameters for the RBF network. The number of processor units in the hidden layer of the network is equal with the number of centers determined by the GCDC method.

Parameters for GCDC:

- initial population size: 400;
- parameters for the fitness function: $\alpha = 1, C = 0.00001$;
- mutation step size: $\sigma = 0.0001$;
- merging radius: $\varepsilon = 0.05$.

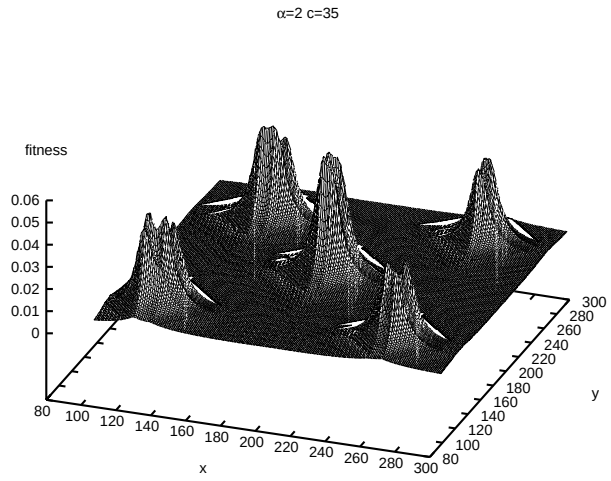


FIGURE 3. Fitness landscape for $\alpha = 2, C = 35$

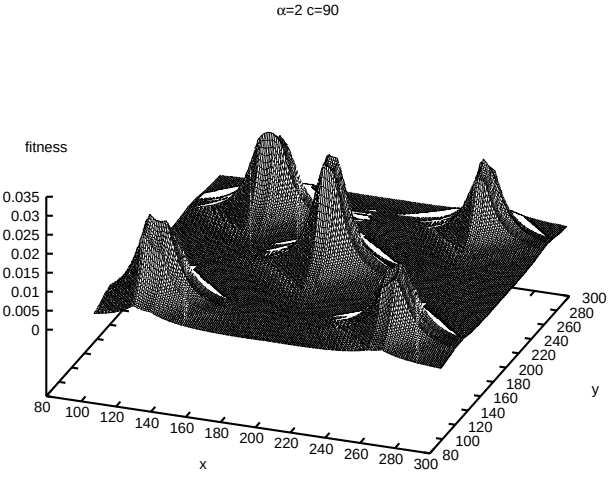


FIGURE 4. Fitness landscape for $\alpha = 2, C = 90$

Gaussian activation functions are used. The parameters for the learning algorithm are:

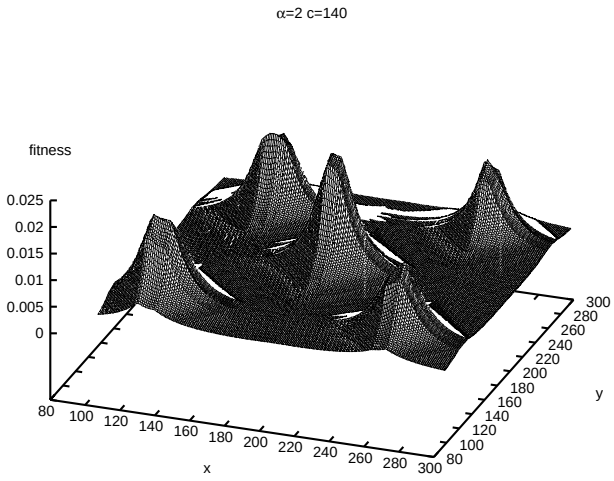


FIGURE 5. Fitness landscape for $\alpha = 2, C = 140$

- learning rate: 0.1;
- maximum number of learning epochs: 10000.

The *generalization error* is calculated using $M = 400$ inputs (that do not belong to the training set) from the interval [0,9.5]. The following formula is used:

$$E_g = \frac{1}{M} \sum_{i=1}^{M} (z_i - y_i)^2 ,$$

where z_i is the expected output and y_i is the network output.

3.2. **RBF networks obtained by using GCDC.** RBF network has been trained using 10 data sets. Each training set consists of 200 points from the interval [0,9.5]. In each set the points are organized in 50 well-separated clusters. For each set the GCDC method is performed and RBF neural network topologies are created based on the returned results.

In 5 cases the number of centers determined by GCDC is 50. In other 5 cases there is a little difference (maximum +4). For some classes more centers are considered. These differences have only minor effects on the network topologies. There is no situation where the number of clusters determined by GCDC is less than 50 (the optimal number of clusters).

After training the obtained RBF networks, the mean generalization error is 0.539953496. Satisfactory approximation results are obtained (Figure 6).

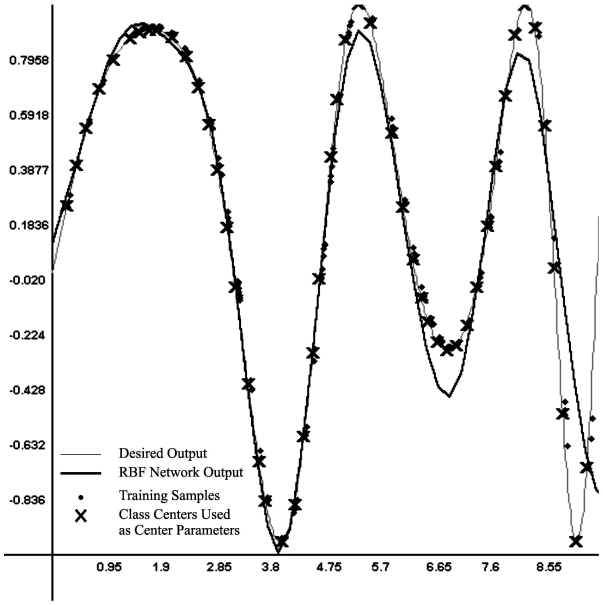


FIGURE 6. 200 training samples organized in 50 clusters, centers determined by the GCDC technique, output of the RBF network after 10000 training epochs.

3.3. **RBF networks obtained by using randomly generated centers.** A training set of 200 points organized in 20 clusters is considered. 20 centers are randomly selected from this set. The RBF network is designed using these centers. The procedure is repeated 10 times. After training the obtained RBF networks the mean generalization error is 0.634810589.

The GCDC technique is performed for clustering the same data set. The method finds 20 clusters and corresponding centers. Based on the returned results a RBF neural network is designed. After 10000 learning epochs the 0.591574517 generalization error is achieved. Better result is obtained using GCDC than using randomly selected centers.

4. Experiment 3

A RBF Neural Network is used for approximating the function:

$$f : [0,1] \to R, f(x) = \left(x - \frac{1}{3}\right)^3 \cdot \frac{1}{27}.$$

The GCDC technique is compared with the generalized k-means algorithm.

4.1. **Experimental Conditions.** A training set consisting of 100 data points organized in 18 clusters is considered.

For k-means algorithm the number of centers is randomly generated in the range 10-25 (we assume that there are more than 10 and less than 25 clusters). 10 tests with 10 different values for the number of centers are performed.

The parameters for the GCDC algorithm are:

-initial population size: 200;

-parameters for the fitness function: $\alpha = 1, C = 0.00001$;

-mutation step size: $\sigma = 0.00001$;

-merging radius: $\varepsilon = 0.02$.

The learning rate for the training process is fixed to 0.1. The learning process will stop if the 0.00005 global learning error is achieved.

The generalization error is calculated using $M = 400$ inputs from the interval $[0, 1]$.

4.2. **Obtained Results and Conclusions.** The results obtained using the k-means algorithm are presented in Table 1. The mean generalization error is: 0.002228871.

GCDC detects 18 clusters and corresponding centers (Figure 7). Using these 18 centers for designing the RBF neural network the learning error of 0.00005 is achieved in 10945 epoches. The generalization error is 3.442700794496429E-4.

A better result is obtained using GCDC than using k-means. The method is able to determine the optimal number of the centers. Using the k-means method much better result is obtained by using 18 or greater value for the number of centers, than using 17 or a smaller value (18 was the real number of the centers). The learning process is thus very sensitive to the number of clusters.

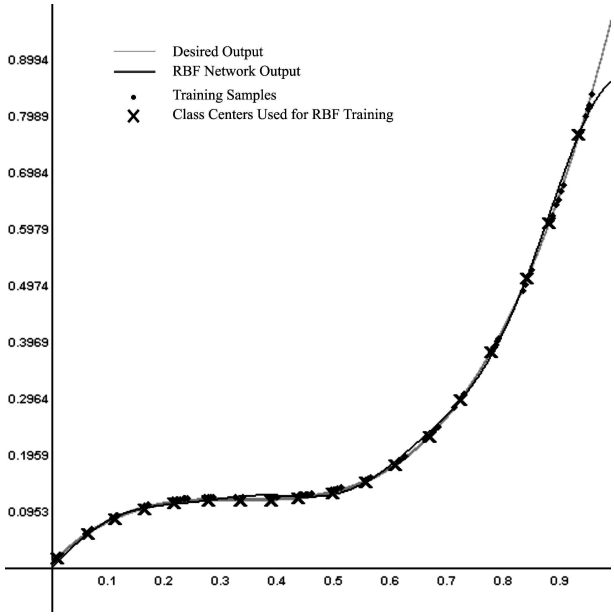


FIGURE 7. 100 training samples organized in 18 clusters, centers determined by the GCDC technique, output of the RBF network after 10945 training epochs.

5. Conclusions

Based on the GC metaheuristics, GCDC is a new evolutionary technique for dynamic clustering. Experimental results indicate that GCDC could be a powerful instrument for data clustering.

The use of GCDC for designing optimal RBF neural network topologies is investigated. Better results are obtained than using standard methods.

References

[1] Broomhead D. S., Lowe D.; *Multivariable Functional Interpolation and Adaptive Networks*, Complex Systems, 2 (1988), pp. 321-355.

[2] Dumitrescu D.; *Algoritmi Genetici şi Strategii Evolutive - Aplicaţii în Inteligenţa Artificială şi în Domenii Conexe*, Editura Albastra, Cluj Napoca, 2000.

[3] Dumitrescu D., Lazzerini B., Jain L. C., Dumitrescu A.; *Evolutionary Computation*, CRC Press, Boca Raton, 2000.

No. of Centers	No. of Epoches	Generalization Error
10	42386	0.003929447755582894
11	26312	0.0039125335843709025
12	15889	0.0038635588999552293
14	8218	0.0037191067346458145
16	2153	0.0028095882895919533
17	2479	0.002400189413222201
18	5466	7.485072155731134E-4
19	10208	5.057298901372404E-4
20	10017	2.240292093213028E-4
23	4918	1.76023288279397E-4

TABLE 1. Generalization errors obtained in 10 runs using the generalized k-means algorithm and 10 different values for the number of centers

[4] Dumitrescu D.; *Genetic Chromodynamics*, Studia Univ. Babes-Bolyai, Ser. Informatica, 35 (2000), pp. 39-50.

[5] Dumitrescu D.; *A New Evolutionary Method and its Applications in Clustering*, Babeş-Bolyai University, Seminar on Computer Science,2 (1998), pp. 127-134.

[6] Dumitrescu D., Simon K.; *Evolutionary Clustering Techniques for Designing RBF Neural Networks*, Babeş-Bolyai University, Seminar on Computer Science, (2003).

[7] Dumitrescu D., Simon K.; *Reducing Complexity of RBF Neural Networks by Dynamic Evolutionary Clustering Techniques*, Proceedings of the 11^th Conference on Applied and Industrial Mathematics, (2003).

[8] Dumitrescu D., Simon K.; *Genetic Chromodynamics for Designing RBF Neural Networks*, Proceedings of SYNASC, (2003).

[9] Dumitrescu D., Simon K.; *A New Dynamic Evolutionary Clustering Technique. Application in Designing RBF Neural Network Topologies. I. Clustering Algorithm*, Studia Univ. Babes-Bolyai, Ser. Informatica, (2004).

[10] Enăchescu C.; *Caracterizarea Reţelelor Neuronale ca şi Metode de Aproximare- Interpolare*, Petru Maior University, Buletinul Stiintific, 7 (1994).

[11] Enăchescu C.; *Elemente de Inteligenţă Artificială*, Petru Maior University, Tg. Mureş, 1997.

[12] Haykin S.; *Neural Networks*, Macmillan College Publishing Company, New York, 1994.

[13] Moody J., Darken C.; *Fast Learning in Networks of Locally Tuned Processing Units*, Neural Computation, 1 (1989), pp. 281-294.

[14] Oltean M., Groşan C.; *Genetic Chromodynamics Evolving Micropopulations*, Studia Univ. Babes-Bolyai, Ser. Informatica, (2000).

[15] Poggio T., Girosi F.; *Networs for Approximation and Learning*, Proceedings of IEEE, 78 (1990), pp. 1481-1497.

[16] Powell M. J. D.; *Radial Basis Functions for Multivariable Interpolation: A review*, in Algorithms for Approximation, J. C. Mason and M. G. Cox, ed., Clarendon Press, Oxford, 1987, pp. 143-167.

[17] Schreiber T.; *A Voronoi Diagram Based Adaptive k-means Type Clustering Algorithm for Multidimensional Weighted Data*, Universitat Kaiserslautern, Technical Report, (1989).

[18] Selim S. Z., Ismail M. A.; *k-means Type Algorithms: A Generalized Convergence Theorem and Characterization of Local Optimality*, IEEE Tran. Pattern Anal. Mach. Intelligence, PAMI-6, 1 (1986), pp. 81-87.

[19] Simon K.; *OOP Pentru Calculul Neuronal*, Petru Maior University, Dipl. Thesis, 2002.

[20] Simon K.; *Evolutionary Clustering for Designing RBF Neural Networks*, Babeş-Bolyai University, MSc. Thesis, 2003.

"Babeş-Bolyai" University, Faculty of Mathematics and Computer Science, Computer Science Department, Cluj Napoca, Romania

E-mail address: ksimon@nessie.cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume L, Number 1, 2005

A PROGRAMMING INTERFACE FOR NON-HIERACHICAL CLUSTERING

GABRIELA ŞERBAN

Abstract. Clustering is one of the important techniques in Data Mining. Clustering methods aim at grouping objects into clusters, so that the objects within a cluster are more similar to each other than objects in different clusters. The similarity between two objects is defined by a distance function. Clustering techniques are used in a variety of domains like: Natural Language Processing, Databases, HealthCare. In this paper we present a new programming interface for non-hierarchical clustering. Using this interface, we can simply develop non-hierarchical clustering applications. Using the designed interface, we made an experiment for words clustering, using a Romanian corpus.

Keywords: Clustering, Programming, Interface.

1. Introduction

The purpose of this paper is to present a standard interface for programming clustering tasks. The interface is meant to facilitate the development of software for clustering in different domains. In particular, the interface should facilitate an approach in which objects to be clustered and attributes describing the objects can be designed and implemented separately and then interconnected relatively easily in a standard, uniform fashion.

The aim of the proposed approach is to abstract the clustering issue, assuring a general approach, independent of the concrete representations of the entities involved in the clustering process.

In various domains clearly appears the necessity of clustering different kind of objects, with respect to different kind of attributes.

For example, in the field of Natural Language Processing we often need to group words by the similarity of their meanings, using a given corpus of words. In this

Received by the editors: March 20, 2005.

2000 *Mathematics Subject Classification.* 68N19, 62H30.

1998 *CR Categories and Descriptors.* I.5.3[**Computing Methodologies**]: Pattern Recognition – *Clustering*; D.1.5[**Software**]: Programming Techniques – *Object-Oriented Programming*;

example, the objects to be clustered are words and the attributes characterizing the objects are also words (from the corpus) [2].

Another example in which clustering is needed is HealthCare. HealthCare rises the problem of grouping patients in classes (clusters) with respect to the values of a number of symptoms for a given disease. In this kind of problems, the objects to be clustered are patients, and the attributes are symptoms [9].

There are many other domains in which clustering is needed, but for different kinds of objects and attributes.

That is why, in this paper we propose an unitary approach for all the clustering applications, independent of the type of objects to be clustered and the type of attributes characterizing the clustering process.

2. Clustering

As it is well-known, clustering is a partition of data into groups of similar objects.

Let us consider the following issue: given n objects $O_1, O_2, ..., O_n$, and m attributes $A_1, A_2, ...A_m$ (a set of relevant characteristics of the analyzed objects) ,we intend to group the objects in a given number k of *clusters*, so that the objects within a cluster are more similar (related to the given attributes) to each other than objects in different clusters.

For computing the similarities between objects we use the vector-space model, which means that the vector $\vec{O_i} = ({O_i}^1, {O_i}^2, ..., {O_i}^m)$ is associated with an object O_i as following: ${O_i}^j$ is a real number that gives a classification of the object O_i from the point of view of attribute A_j.

For designing our interface, the computation method of ${O_i}^j$ is unimportant.

Similarity and dissimilarity between objects are calculated using metric or semi-metric functions, applied to the attribute values characterizing the objects.

There are several methods for computing the similarity between two objects represented by their associated vectors (as defined above).

(1) The similarity measure between two objects O_a and O_b is defined as the *normalised cosine* between the vectors $\vec{O_a}$ and $\vec{O_b}$ [7]:

$$sim(\vec{O_a}, \vec{O_b}) = cos(\vec{O_a}, \vec{O_b}) = \frac{\sum_{j=1}^{m} O_a^j \times O_b^j}{\sqrt{\sum_{j=1}^{m} {O_a^j}^2} \times \sqrt{\sum_{j=1}^{m} {O_b^j}^2}}.$$

(2) the similarity measure between two objects O_a and O_b is defined as

$$sim(\vec{O_a}, \vec{O_b}) = \frac{1}{\sum_{j=1}^{m} \left(O_a^j - O_b^j\right)^2}.$$

(3) the similarity measure between two objects O_a and O_b is defined as

$$sim(\vec{O_a}, \vec{O_b}) = \frac{1}{\sum_{j=1}^{m} |O_a^j - O_b^j|}.$$

The *distance* between two objects O_a and O_b is defined as

$$d(\vec{O_a}, \vec{O_b}) = \frac{1}{sim(\vec{O_a}, \vec{O_b})}.$$

A well-known class of clustering methods is the one of the partitioning methods, with representatives such as the *k-means* algorithm. Essentially, given a set of n objects and a number $k, k \leq n$, such a method divides the object set into k distinct and non-empty partitions. The partitioning process is iterative and heuristic; it stops when a "good" partitioning is achieved. A partitioning is "good", as we said, when the intra-cluster similarities are high and inter-cluster similarities are low.

We give next the non-hierarchical clustering algorithm (k-means algorithm) [8].

`Algorithm k-means is`
Input: `-` `The set` $X = \{\vec{O_1}, \vec{O_2}, \cdots, \vec{O_n}\}$ `of n vector objects to be clusterised,`
`-` `the distance measure` $d : R^m \times R^m \to R$, `between objects in a multi-dimensional space,`
`-` `k, the number of desired clusters,`
`-` `a function for computing the mean of a cluster` C, $\mu : C \to R$,
`-` `the coefficient` σ `(the threshold).`
Output: `-` `the set of clusters` $C = \{C_1, C_2, \cdots, C_k\}$.
Begin
`Select` k `initial centroids` $\{\vec{f_1}, \vec{f_2}, \cdots, \vec{f_k}\}$
`While the diameter of a cluster` $\geq \sigma$ `do`
`For all clusters` $C_j \in C$ `do`
$C_j = \{\vec{O_i} \mid \forall \vec{f_l}\ d(\vec{O_i}, \vec{f_j}) \leq d(\vec{O_i}, \vec{f_l})\}$
`EndFor`
`For all clusters` $C_j \in C$ `do`
$\vec{f_j} = \vec{\mu}(C_j)$
`EndFor`
`EndWhile`
End.

As distance measure we considered:

$$d(\vec{O_a}, \vec{O_b}) = \frac{1}{sim(\vec{O_a}, \vec{O_b})}$$

and as centroid the mean of the cluster:

$$\vec{\mu}(C_j) = \frac{1}{\mid C_j \mid} \sum_{\vec{O} \in C_j} \vec{}$$

We define the diameter of a cluster as *the distance between the least similar elements in a cluster*.

We also mention that the algorithm stops when the diameter of each cluster is less then a fixed threshold.

3. The programming interface

In this section we propose a standard interface that allows a simple development of clustering applications, providing a uniform development for all kind of applications.

The programming interface provides a hierarchy of classes and interfaces that can be used in all clustering applications. The clustering mechanism will be the same for all types of objects and attributes.

The interface is realized in JDK 1.5, and is meant to facilitate software development for non-hierarchical clustering.

There are three basic entities (objects): *objects to be clustered*, *attributes* (that characterize the objects) and *clustering*.

For designing the interface, we made an abstraction of the clustering mechanism, in order to be used for any kind of data (objects and attributes). Much more, the objects to be clustered are completely separated from the attributes that characterize them (an object has to know nothing about an attribute). Thus, we can easily change the attributes characterizing the objects, without affecting the clustering process.

The *clustering* object is the main object of the interface that manages the clusterization of the given objects related to the given attributes. The *clustering* object provides the behavior specific to the clustering process.

A main characteristic is that the *clustering* object is completely separated from the objects to be clustered and the attributes characterizing the objects (the *clustering* object knows only the behavior provided by the methods from the interface of these entities).

The interface provides an object as a manager for the list of objects to be clustered that manages the creation of the list of objects, attributes and centroids from an external device (file, database).

For using the interface, the user has to define the specialized object classes `CConcreteObject` (the concrete object to be clustered), `CConcreteAttribute`

(the concrete attribute) and `CConcreteManager` (the concrete manager for the list of objects and attributes), by creating instances for each. The list of objects, attributes and centroids given by the manager are then passed to a clustering object (`CClustering`), that will initialize the clustering process and will manage the results.

In the following we present the skeleton of a clustering application.

(1) First, the user defines the class corresponding to the concrete object to be clustered.
public class `CConcreteObject` **extends** `CObjectToBeClusterized`
{...}
(2) Second, the user defines the class corresponding to the concrete attribute that characterizes the objects.
public class `CConcreteAttribute` **implements** `IAttribute`
{...}
(3) The user defines the class corresponding to the concrete manager of objects to be clustered and attributes.
public class `CConcreteManager` **implements** `IClusteringManager`
{
public CListOfObjectsToBeClustered createListOfObjects(){...}
public CListOfAttributes createListOfAttributes(){...}
public CCluster createCentroids(){...}
{...} }

The application class which initializes the clusterization with the data provided by the concrete manager object is always the same (remains unchanged for all non-hierarchical clustering issues), and is described below.

class *Application* {
private Application(){
CConcreteManager cm=new CConcreteManager();
//the manager provides the list of objects to be clusterized
CListOfOjectsToBeClustered l=cm.createListOfObjects();

//the manager provides the list of attributes corresponding to the objects
CListOfAttributes y=cm.createListOfAttributes();

//the manager provides the initial centroids of the clusters
CCluster f=cm.createCentroids();

//the manager initializes the clustering process
CClustering l=new CClustering(l, y, f); }

```
public static void main(String args[]){
        Application apl=new Application();

}}
```

Figure 1 shows a simplified UML diagram of the interface, illustrating the hierarchy of classes. It is important to mention that all the classes provided by the interface, except the concrete classes, remain unchanged for all kinds of clustering applications.

4. The Design of the Interface

The classes used for realizing the interface are the following:

- `IList` **INTERFACE**
  Defines the structure of a list of objects, having operations for managing the list: adding an element on a given position, removing an element from a given position, returning the number of elements from the list, returning an element from a given position.
- `IClusteringManager` **INTERFACE**
  Defines the structure of a manager for the clustering process. The manager provides methods for obtaining the elements that are needed in the clustering process: creating the list of objects, the list of attributes and the initial centroids.
- `CLine`
  Defines the structure for the vector $\vec{O}$ corresponding for an object O (as we had defined above). An element of this vector is a real number, representing a characteristic measure for the object related to an attribute (in this class, the type of an attribute is unimportant). The main methods of this class are for: adding, updating, removing elements, for calculating the similarity between two lines.
- `CCluster`
  In our design a cluster is represented as a list of `CLine`(a line identifies in fact an object). For a cluster, the type of the object is unimportant. The main methods of this class are for: adding, updating, removing elements, for calculating the centroid of a cluster, for testing the equality of two clusters.

OBJECTS

An object is the entity to be clustered.

- `CObjectToBeClustered` **ABSTRACT CLASS**
  Is the basic class for all the objects. The specific objects will be instances of subclasses derived from `CObjectToBeClustered`. An object is identified by its vector representation. The methods of this class

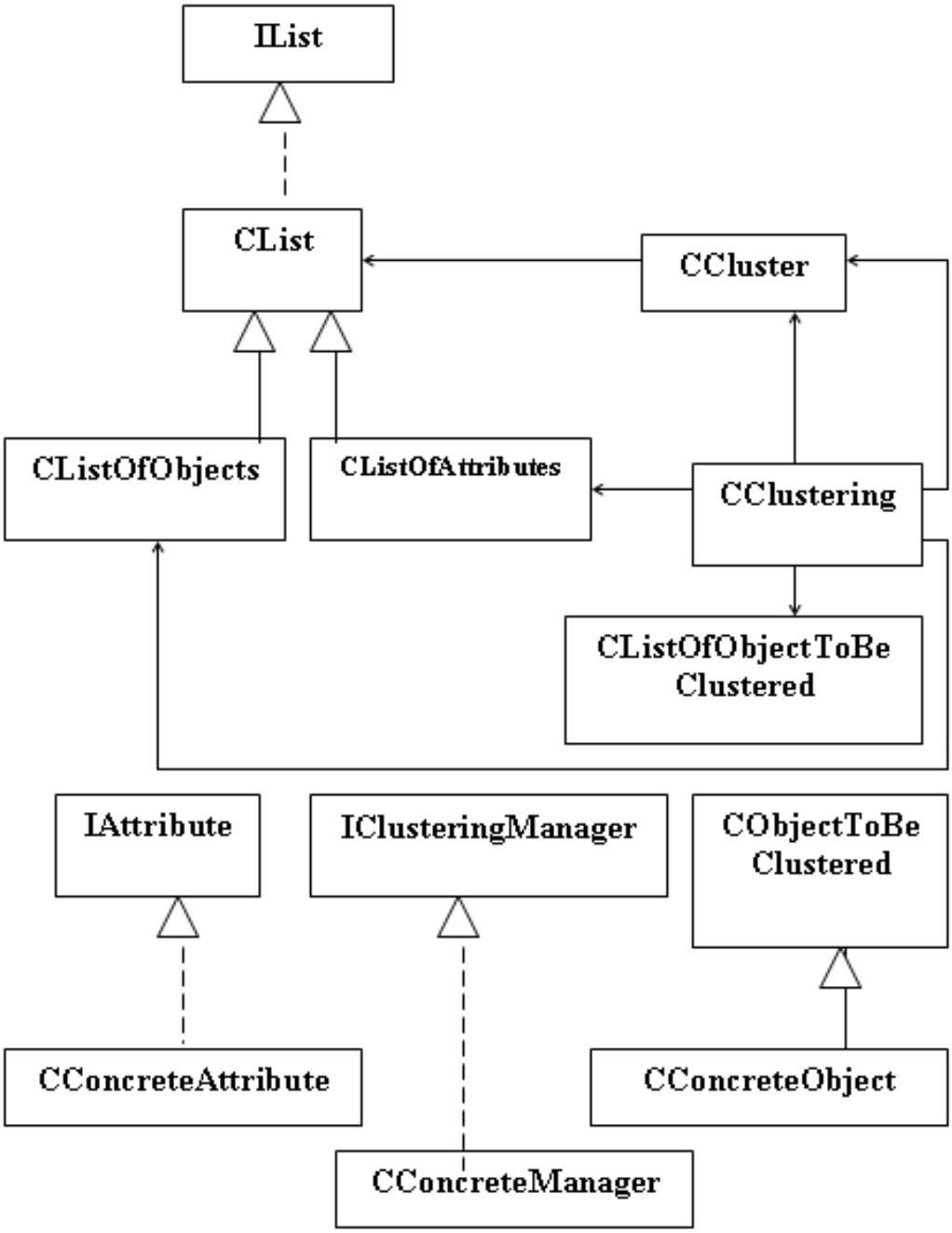


FIGURE 1. The diagram of the programming interface

are for: returning a String with the representation of an object, returning the value of an object, comparing two objects, returning the vector corresponding to an object.

- `CListOfObjects` **ABSTRACT CLASS**
  This class represents the list of objects to be clustered. The main methods are for managing the list.
- `CListOfObjectsToBeClustered` **ABSTRACT CLASS**
  This class maintains a list of objects `CListOfObjects` and the list of vector representations corresponding to each object from the list. The main methods are for managing the components.

ATTRIBUTES

- `IAttribute` **INTERFACE**
  Defines the structure of an attribute characterizing an object to be clustered. The methods of this class are for: returning a String with the representation of an attribute, returning the value of an attribute, comparing two attributes.

CLUSTERING

- `CClustering`
  Is the basic object of the interface, that manages the clustering process of the objects (related to the attributes). Defines the *heart* of the interface, the uniform usage that all objects and attributes are meant to conform to.

  An instance of the clustering class is associated with an instance of a list of objects and a list of attributes at the creation moment. This is made in the constructor of the class `CClustering`. The main method of this class is the method that manages the clusterization process (using the non-hierarchical clustering algorithm) and returns the clusters obtained.

  public class `CClustering`
  {
  private CListOfAttributes y; //reference to the list of attributes
  private CListOfObjectsToBeClusterized l; //reference to the list of objects to be clustered
  private CCluster f; //reference to the initial centroids
  ...
  }

5. Experiments

In order to test the above defined interface, we considered a NLP experiment for the Romanian language. The aim was to clusterize a set of words (to group the words after the similarity of their meanings).

In our experiment, the attributes for the words to be clusterised were also words. Using a corpus for Romanian language, the vector words are computed based on the idea described in [2].

For testing the generality of our interface, we have also developed a clustering application for HealthCare. We mention that all data were taken from the website at [10].

The objects to be clusterized in this experiment were patients: each patient is identified by 9 attributes [9]. The attributes have been used to represent instances. Each instance has one of two possible classes: benign or malignant.

For each experiment, we have defined the classes that provide the current clustering focus, `CConcreteManager`, `CConcreteAttribute` and `CConcreteObject` and the applications for clustering were easily developed.

As a conclusion of our experiments, we have to mention, from a programmer point of view, the advantages of using the above proposed interface:

- is very simple to use;
- the effort for developing a clustering application is reduced – we need to define only three classes, the rest is provided by the interface;
- the user of the interface has to know nothing about the method for clustering the objects, because is provided by the interface;
- we can dynamically change the type of objects to be clustered or the type of the attributes that characterize the objects, and the interface remains unchanged.

6. Conclusions and Further Work

As a conclusion, we have developed a small framework that will help programmers to build, dynamically, their own clustering applications without dealing with the clustering mechanism (that remains unchanged and is provided by the interface). For a concrete application, the programmer has only to create three classes (derived from the classes defined by the interface): a class corresponding to the object to be clustered, a class corresponding to the attribute characterizing an object and, finally, a class corresponding to the entity that manages the objects and attributes.

After defining the concrete classes, the clustering will be made by creating an instance of the class `CClustering` provided by the interface. So, the programmer's effort for developing an application is small.

We mention that using the proposed interface we can simply develop clustering applications for different kind of data (objects). The objects can be words (in

NLP), databases, even patients (in HealthCare), or any other objects for which clustering techniques can be applied.

Further works can be done in the following directions:

- how can the interface be generalized in order to be used both for hierarchical [2] and non-hierarchical clustering;
- how can the interface be generalized for adaptive clustering (there are new Objects to be clustered or/and new Attributes that characterize the already clustered Objects).

References

[1] Jain, A., Dubes, R, "Algorithms for Clustering Data", Prentice Hall, Englewood Cliffs, New Jersey, 1998.

[2] Tatar, D., Serban, G.: "Words Clustering in Question Answering Systems", Studia Universitatis "Babes-Bolyai", Informatica, XLVIII(1), 2003, pp.23–32.

[3] I. Dagan, L. Lee, F. C. N. Pereira: "Similarity-based models of Word Coocurences Probabilities", Machine Learning Journal 34(1–3), 1999, pp.1–26.

[4] C. Orasan, D. Tatar, G. Serban, D. Avram, A. Onet: "How to build a QA system in your back-garden: application to Romanian", EACL '03, Budapest, April 2003, 12-14, pp.139–142.

[5] P. Resnik: "Semantic Similarity in a Taxonomy: An information-Based Measure and its Application to Problems of Ambiguity in Natural language", Journal of AI Research, 1998, Center for the Study of Language and Information (CSLI), pp.1–28 .

[6] G. Serban, D. Tatar, "Word Sense Disambiguation for Untagged Corpus: Application to Romanian Language", Proceedings of CICLing 2003 (Intelligent Text Processing and Computational Linguistics), Mexico City, Mexic, Lecture Notes in Computer Science N 2588, Springer-Verlag, 2003, pp.270-275.

[7] D. Jurafsky, J. Martin: "Speech and language processing", Prentice Hall, 2000.

[8] C. Manning, H. Schutze: "Foundation of statistical natural language processing", MIT, 1999.

[9] Wolberg, W., Mangasarian, O.L.: "Multisurface method of pattern separation for medical diagnosis applied to breast cytology", Proceedings of the National Academy of Sciences, U.S.A., Volume 87, December 1990, pp 9193–9196.

[10] http://www.cormactech.com/neunet, "Discover the Patterns in Your Data", CorMac Technologies Inc, Canada.

Babeş-Bolyai University, Faculty of Mathematics and Computer Science, Cluj-Napoca, Romania

E-mail address: gabis@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume L, Number 1, 2005

FINE-GRAINED MACROFLOW GRANULARITY IN CONGESTION CONTROL MANAGEMENT

DARIUS VASILE BUFNEA, ALINA CAMPAN, AND ADRIAN SERGIU DARABANT

Abstract. A recent approach in Internet congestion control suggests collaboration between sets of streams that should share network resources and learn from each other about the state of the network. Currently such a set of collaborating streams - a macroflow - is organized on host pair basis. We propose in this paper a new method for grouping streams into macroflows when they behave similarly. A flow behavior is described by a set of state variables, such as the round trip time, retransmission time out or congestion window size. This extended macroflow granularity can be used in an improved Congestion Manager.

1. Introduction

Congestion control aims to control and adapt the transmission rate of the Internet streams in order to reduce the amount of dropped packets in case of overloaded communication lines and routers. Practical congestion control approaches work either at *protocol level* or at *router level*. A *transport protocol* should normally implement a congestion control algorithm. The TCP protocol, which transports over 90% of Internet data, treats this aspect. But there are other protocols, which remain congestion unaware. *Routers* have their congestion control policies and algorithms for handling congestion situations that are usually induced by misbehaved congestion unaware flows. The two mentioned approaches do not exclude each other; rather they are completing each other.

In order to properly offer reliable data transmission and congestion control, a TCP connection uses some state variables such as: the round trip time (*rtt*), the

Received by the editors: March 15, 2005.

2000 *Mathematics Subject Classification.* 62H30, 90B20, 68U35.

1998 *CR Categories and Descriptors.* 62H30 [**Statistics**]: Multivariate analysis – *Classification and discrimination; cluster analysis*; 90B20 [**Operations research, mathematical programming**]: Operations research and management science – *Traffic problems*; 68U35 [**Computer science**]: Computing methodologies and applications – *Information systems (hypertext navigation, interfaces, decision support, etc.).*

retransmission time out (*rto*) [4], the congestion window (*cwnd*) and the slow start threshold (*sstresh*) [1]. Usually, each TCP connection maintains independently its own state variables and performs its own calculation for determining these variablesŠ values.

But even when each stream, independently, incorporates congestion aware algorithms, a set of concurrent streams will still compete with each other for network resources, rather than share them effectively [2]. Recent approaches introduce the idea of Internet streams *collaborating* for an improved congestion control mechanism. Rigorous delimited (defined) set of streams should share network resources and learn from each other about the state of the network. Currently, such a set of collaborating streams, referred as a *macroflow*, is organized on host pair basis; i.e. a macroflow comprises connections sharing the same (source IP, destination IP) pair. We propose in this paper a new method for grouping streams into macroflows according to their similar behavior. This method provides an accurate, less naive approach for delimiting macroflows inside the overall set of connections maintained by a host. As a consequence, more connections will be detected as being part in one macroflow and will share their network knowledge. This approach is meant to be part of an improved congestion Control Manager.

1.1. **Related Work.** Floyd suggested in [6] that the *rtt* and *rto* values should be the same for all connections that share the same (source IP, destination IP) pair in the same moment in time. For this reason, she claimed that the network level should be maintaining the values of these state variables, and not the transport level. However, Floyd did not further explore this approach.

[5] joins the idea of sharing state variables between flows, on host pair basis. In addition, she gives practical suggestions and solutions for accomplishing this, in certain concrete situations.

[3] describes the LINUX caching mechanism of state variables values. One set of information is maintained for each destination IP. The cached values serve for state variables initialization of new connections targeting the same destination IP. Thus, the LINUX caching mechanism also functions on host pair basis.

A state of the art approach [2] in congestion control suggests a practical way for the collaboration between transport protocols and applications. This collaboration should take place into an integrated *Congestion Manager* (CM) framework. All protocols and applications involved in such a framework should provide their network knowledge (*rtt*, packet losses) to the CM. The CM should aggregate all these information on host pair basis (macroflow basis), ŞlearnŤ from them and inform the protocols and applications, in a synchronous or asynchronous manner, about when and how much data they can safely Şput on the wireŤ. Practically,

the collaboration will take place, mediated by the CM, between connections inside a macroflow; no collaboration will happen across macroflows. So, more adequate the macroflows are established, more efficient the CMŠs control will be.

1.2. **Contributions.** We propose in this paper a new method for grouping streams into macroflows when they behave similarly. A flow behavior is defined by a set of state variables, such as the round trip time, retransmission time out or congestion window size. The advantage is that we can cluster together streams not only on host pair basis, but also on LAN pair basis; even more generally, streams sharing a particular network bottleneck will be identified by our method. This extended macroflow granularity can be used in an improved Congestion Manager.

2. Data Model

2.1. ***Rtt* Vectors.** We consider the case of an upload server that treats a high number of simultaneous incoming connections. The aim is to establish (infer) inside this set of connections some groups of connections with similar behavior. A Congestion Manager running on that server will treat such a group as a macroflow.

We denote by S the *server machine* itself or its *network identification IP address*.

Each incoming connection is identified by the server S by a pair $(C_{IP} : C_{port})$, where C_{IP} is the client's IP address and C_{port} is the client's port identification.

During the life time of each $(C_{IP} : C_{port})$ connection, the server S will periodically measure and retain the values of some state variables, such as the round trip time, retransmission time out or congestion window size. We based our experiments on measurements of the round trip time (*rtt*) state variable. Practical ways for the achievement of measurements are described in [8, 9].

Therefore, from the point of view of the upload server S, the incoming connection $f = (C_{IP} : C_{port})$ during the time interval (t_b, t_e) is described by the *rtt vector* $V = (r_1, r_2, \ldots, r_k)$ where:

- $(t_b, t_e) \subseteq (C_{IP} : C_{port})$ connection life time;
- Δt is the interval between two consecutive measurements;
- $k = (t_e - t_b)/\Delta t$;
- r_i is the *rtt* value measured at the $t_b + \Delta t * (i - 1)$ time moment.

We say that the *rtt vector* associated to a connection describes the connection's behavior. The choice of the *rtt* state variable for describing a connection behavior is justified as follows. For two connections f_1 and f_2 coming from the same client or LAN the *rtt* values measured at the same moment in time are quasi-identical. Therefore, their associated *rtt vectors* during the same time interval are

also quasi-identical. This means that f_1 and f_2 manifest a similar behavior, which justifies their placement in the same macroflow. As we said, we want to extend the macroflow granularity outside the host-pair scope, on the basis of similar behavior; but we also want to keep such connections (as f_1 and f_2 are) as much as possible together in macroflows. So, according to our modeling such connections must have similar behavior. The rtt vectors model ensures that fact.

2.2. **Similarity Measure.** For grouping similarly behaving flows we will use, as described in the next paragraph, an artificial intelligence clustering algorithm. Such an algorithm needs a similarity measure and a distance function for comparing and differentiating two analyzed flows. We propose next such measures and justify our choice.

We associated to a connection an *rtt vector* describing its behavior. The *rtt vector* reflects the *rtt* temporal evolution of that flow. Two connections will be considered more similar as they are more linearly correlated (e.g. the *rtt* values for the two connections increase and decrease at the same moments in time). A statistical measure for the linear correlation of two vectors is the *Pearson coefficient*.

Given two connections, $f_1 = (C^1_{IP} : C^1_{port})$ and $f_2 = (C^2_{IP} : C^2_{port})$ measured during the time interval (t_b, t_e) and their associated *rtt vectors* $V_1 = (r_{11}, r_{12}, \ldots, r_{1k})$ and $V_2 = (r_{21}, r_{22}, \ldots, r_{2k})$, the *Pearson correlation coefficient* of f_1 and f_2 is defined as:

$$P(V_1,V_2) = \frac{\sum_{i=1}^{k} r_{1i}\cdot r_{2i} - \frac{\sum_{i=1}^{k} r_{1i}\cdot \sum_{i=1}^{k} r_{2i}}{k}}{\sqrt{\left(\sum_{i=1}^{k} r_{1i}^2 - \frac{(\sum_{i=1}^{k} r_{1i})^2}{k}\right)\cdot\left(\sum_{i=1}^{k} r_{2i}^2 - \frac{(\sum_{i=1}^{k} r_{2i})^2}{k}\right)}}$$

$$= \frac{\sum_{i=1}^{k}(r_{1i}-\overline{r_1})\cdot(r_{2i}-\overline{r_2})}{\sqrt{\left(\sum_{i=1}^{k}(r_{1i}-\overline{r_1})^2\right)\left(\sum_{i=1}^{k}(r_{2i}-\overline{r_2})^2\right)}} \quad (1)$$

where $\overline{r_1}$ and $\overline{r_2}$ are the mean values of V_1 and V_2.

$P(V_1, V_2)$ takes values in [-1,1] interval, a value of 1 meaning that the compared vectors are linearly correlated, and a value of -1 meaning that they are inversely linearly correlated. We chose to transport the Pearson coefficient values in [0,1] interval. However, this will not affect the semantics of the transformed coefficient.

Its maximal value will still indicate o positive correlation between parameters, and the minimal value a negative correlation. Therefore, the similarity measure we use for comparing connections will be:

$$\overline{P}(V_1, V_2) = \frac{P(V_1, V_2) + 1}{2} \tag{2}$$

For differentiating connections we use a distance function defined by:

$$d_P(V_1, V_2) = 1 - \overline{P}(V_1, V_2) \tag{3}$$

d_P take values in [0,1]; two identical flows will be at 0 distance, two negatively correlated flows will be separated by a distance of 1.

The distance function defined on the basis of the correlation coefficient as above does not satisfy the triangle inequality; it is, therefore, what is called a *semi-metric*.

We have to notice a shortcoming of the correlation coefficient in describing vectors with a constant evolution (e.g. vectors with all the components equal). If one of the vectorial arguments of P is constant, the correlation coefficient is undefined. We choused to consider it -1, as nothing can be said about the correlation between such arguments and we want them not to disturb the classification of the other connections with well defined behavior.

3. The *MacroflowIdentification* Algorithm

Let $F = f_1, f_2, \ldots, f_n$ be the set of all incoming concurrent connections served by S. For the (t_b, t_e) time interval, the measured *rtt vectors* are $V = V_1, V_2, \ldots, V_n$, where V_i is the *rtt vector* associated to f_i, $f_i = (C^i_{IP} : C^i_{port}), V_i = (r_{i1}, r_{i2}, \ldots, r_{ik})$.

We use an *agglomerative hierarchical clustering algorithm* for grouping in macroflows the concurrent connections described by similar *rtt vectors*. This bottom-up strategy starts by placing each connection in its own cluster (macroflow) and then merges these atomic clusters into larger and larger clusters (macroflows) until a *termination condition* is satisfied.

At each iteration, the closest two clusters (macroflows) are identified. The distance between two clusters M_i and M_j is considered to be, as defined in (4), the maximum distance of any pair of objects in the cartesian product $M_i \times M_j$. If the distance between these two closest clusters does not exceed a given threshold *thr_max_dist*, we merge them and the algorithm continues by a new iteration. Otherwise, the algorithm stops. So, the *termination condition* holds when there are no more clusters closer than a given threshold.

This decision regarding the termination condition is justified. We want that the resulting macroflows do not contain *any* ŞwrongŤ placed connections, so that the subsequent decisions based on our macroflow delimitation not to be erroneous. A macroflow is ŞcorrectŤ if any pair of its objects are similar enough.

The threshold *thr_max_dist* was chosen above 0.95, to ensure correct macroflow construction. By merging two clusters that are close enough with respect to the threshold *thr_max_dist* ensures that, inside the obtained merged cluster (macroflow), any two connections are not more distant than *thr_max_dist*. So, it is safe to place them into the same macroflow.

```
Algorithm MacroflowIdentification is
Input:  n, the number of concurrent connection at server S;
        F = {f_1, f_2, ..., f_n} the set of concurrent connection at S;
        V = {V_1, V_2, ..., V_n}, V_i = (r_i1, r_i2, ..., r_ik), i = 1..n, the rtt vectors
        associated to the connections;
        thr_max_dist, the maximal distance threshold for two connections
        to be admitted in the same macroflow.
Output:m, the number of macroflows inferred in the concurrent connections
        set;
        M = {M_1, ..., M_m}, the inferred macroflows, where
        M_i ≠ ∅, i = 1..m, ∪_{i=1}^{m} M_i = F, M_i ∩ M_j = ∅, i, j = 1..m, i ≠ j.
Begin
    m := n;
    M := ∅;
    For i:= 1 to m do
       M_i := {f_i};
       M := M ∪ {M_i};
    End For;
    While (m>1) and (Continue(M,thr_max_dist,M_merge1, M_merge2)=true) do
       M_new := M_merge1 ∪ M_merge2;
       M := M − {M_merge1, M_merge2} ∪ {M_new};
       m := m-1;
    End While;
End.


Function Continue(M the set of current macroflows, thr_max_dist,
                  out M_merge1, out M_merge2):boolean is
Begin
    min_dist := ∞;
    For each M_i ∈ M
```

```
        For each M_j ∈ M, M_j ≠ M_i
(4)         dist(M_i, M_j) = max{d_P(v_r, v_t) | f_r ∈ M_i, f_t ∈ M_j};
            If dist(M_i, M_j) < min_dist
                min_dist := dist(M_i, M_j);
                M_merge1 := M_i; M_merge2 := M_j;
            End If;
        End For;
    End For;
    If min_dist < thr_max_dist Return True;
    Else Return False;
    End If;
End Function.
```

Function Continue determines the closest two clusters from the clusters set M. It will return true if these clusters are closer than thr_max_dist and false otherwise.

4. Results and Evaluation

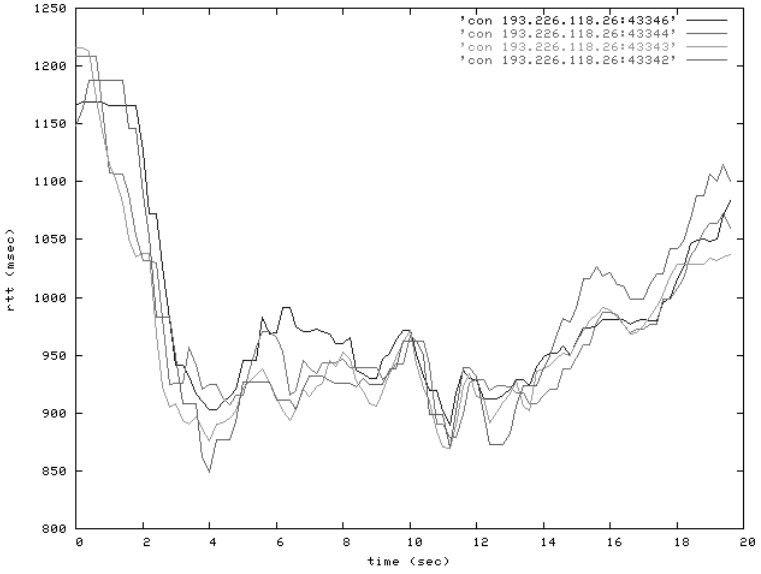


Figure 1. A macroflow composed of connections originating from the same client IP

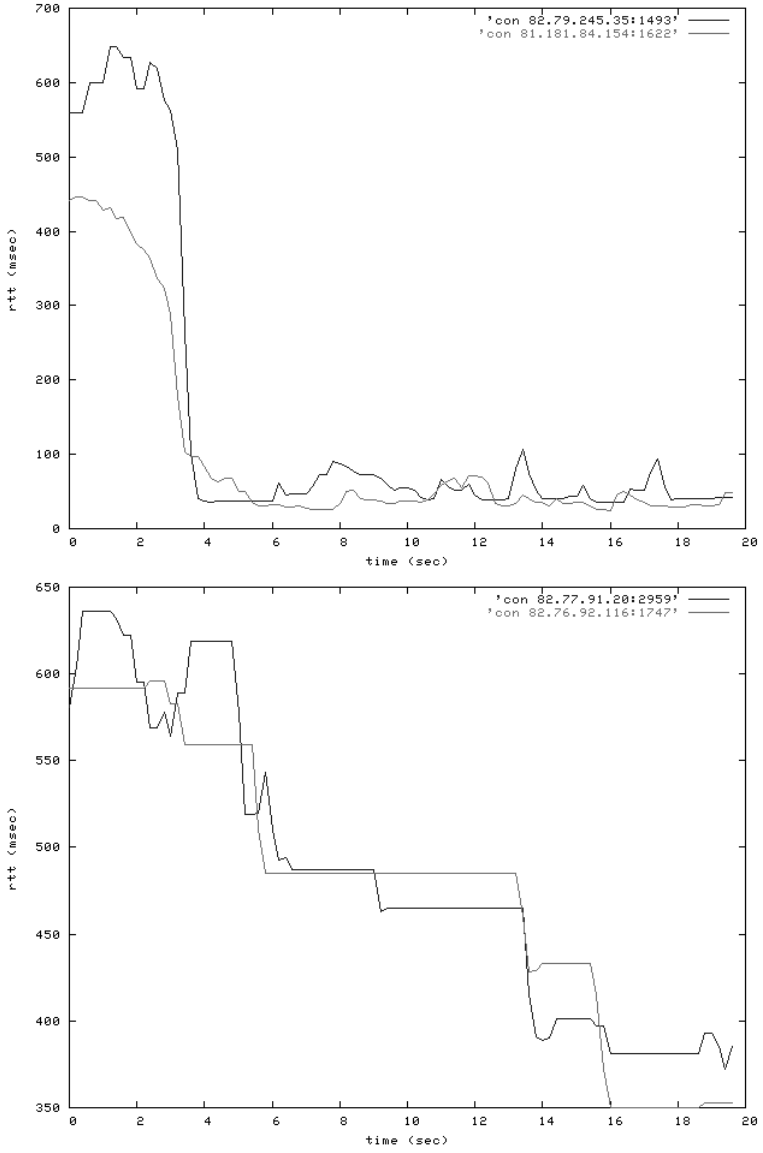


FIGURE 2. Macroflows composed of connections originating from different client IPs

To test the efficiency of the proposed algorithm, we used it on an http upload server, with a high number of incoming connections. We measured the *rtt* state

variable for the incoming connections at S during a larger time interval and we considered for clustering samples of 20 seconds. We take the case of one such sample, composed of 89 connections, originated from 50 different remote hosts. Inside this connection set our algorithm detected 38 macroflows.

The connections originating from the same client were, most of them, clustered together in one or few clusters. Figure 1 represents a macroflow in which were grouped together 4 connections with the same client IP.

But the algorithm also detected, as we intended, macroflows over connections coming from different client IPs. Two such macroflows are illustrated in Figure 2. It can be clearly seen the similar *rtt* evolution during time for the connections of each macroflow. This fact might happen in different situations: client IPs hosted in the same remote LAN or client IPs sharing the same bottleneck toward server S.

The connections with quasi-constant (almost all components of the associated *rtt vectors* are equal) were all grouped together in one cluster - however, taking into account the behavior of the Pearson correlation coefficient in the presence of a constant vector argument they are not to be considered as similar and forming a macroflow.

5. Conclusions and Future Work

We suggested in this paper a data model and an algorithm for extending the macroflow granularity outside of the host-pair approach. Our method will prove its advantages in a Congestion Manager framework.

As a future work we plan to explore the use of different similarity measures and other state variables to compare the timely evolution of the connections being analyzed. We also want to extend the clustering method to an incremental variant having the ability to deal with new connections entering or leaving the system at any given moment.

References

[1] Allman, M., Paxson, V., Stevens, W. - TCP Congestion Control, IETF RFC 2581, April, 1999.

[2] Balakrishnan, H., Seshan, S. - The Congestion Manager, IETF RFC 3124, June 2001.

[3] Sarolahti, P., Kuznetsov, A. - Congestion Control in Linux TCP, In Proc. of the FREENIX Track: 2002 USENIX Annual Technical Conference, pp 49-62, 2003.

[4] Paxon, V., Allman, M. - Computing TCPŠs Retransmission Timer, IEFC RFC 2988, November 2000.

[5] Touch, J. - TCP Control Block Interdependence, IETF RFC 2140, April 1997.

[6] Floyd, S. - A report on Some Recent Developments in TCP Congestion Control, IEEE Communication Magazine, 39(4), 2001.
[7] Han, J., Kamber, M. - Data Mining: Concepts and Techniques, Morgan Kaufmann Publishers, 2001.
[8] Bufnea, D., Sterca, A., Cobarzan, C., Boian, F. - Improving the Round Trip Time Estimation in Internet Routers, In Carpathian Journal of Mathematics, 20(2), Proc of the 4th Intl. Conf. on Applied Mathematics (ICAM4), Baia Mare, Romania, pp 149-154, 2004.
[9] Bufnea, D., Sterca, A., Cobarzan, C., Boian, F. - TCP State Variables Sharing, Proc. of the Symposium Zilele Academice Clujene, Romania, 2004.

Babes-Bolyai University, Cluj-Napoca,Romania
E-mail address: bufny@cs.ubbcluj.ro

Babes-Bolyai University, Cluj-Napoca, Romania
E-mail address: alina@cs.ubbcluj.ro

Babes-Bolyai University, Cluj-Napoca,Romania
E-mail address: dadi@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume L, Number 1, 2005

CORE BASED INCREMENTAL CLUSTERING

GABRIELA ŞERBAN AND ALINA CÂMPAN

Abstract. Clustering is a data mining activity that aims to differentiate groups inside a given set of objects, with respect to a set of relevant attributes of the analyzed objects. Generally, existing clustering methods, such as *k-means* algorithm, start with a known set of objects, measured against a known set of attributes. But there are numerous applications where the attribute set characterizing the objects evolves. We propose in this paper an incremental, *k-means* based clustering method, *Core Based Incremental Clustering (CBIC)*, that is capable to re-partition the objects set,when the attributes set increases. The method starts from the partitioning into clusters that was established by applying *k-means* or *CBIC* before the attribute set changed. The result is reached more efficiently than running *k-means* again from the scratch on the feature-extended object set. Experiments proving the method's efficiency are also reported.

Keywords: Data Mining, clustering, k-means.

1. Introduction

Unsupervised classification, or clustering, as it is more often referred as, is a data mining activity that aims to differentiate groups (classes or clusters) inside a given set of objects. The inferring process is carried out with respect to a set of relevant characteristics or attributes of the analyzed objects. The resulting groups are to be built so that objects within a cluster to have high similarity with each other and low similarity with objects in other groups. Similarity and dissimilarity between objects are calculated using metric or semi-metric functions, applied to the attribute values characterizing the objects.

A large collection of clustering algorithms is available in the literature. [5] and [6] contain comprehensive overviews of existing techniques.

A well-known class of clustering methods is the one of the partitioning methods, with representatives such as the *k-means* algorithm or the *k-medoids* algorithm.

Received by the editors: March 15, 2005.

2000 *Mathematics Subject Classification.* 62H30, 68U35.

1998 *CR Categories and Descriptors.* 62H30 [**Statistics**]: Multivariate analysis – *Classification and discrimination; cluster analysis*; 68U35 [**Computer science**]: Computing methodologies and applications – *Information systems (hypertext navigation, interfaces, decision support, etc.)*;

Essentially, given a set of n objects and a number $k, k \leq n$, such a method divides the object set into k distinct and non-empty partitions. The partitioning process is iterative and heuristic; it stops when a "good" partitioning is achieved. A partitioning is "good", as we said, when the intra-cluster similarities are high and inter-cluster similarities are low.

Generally, these methods start with a known set of objects, measured against a known set of attributes. But there are numerous applications where the object set is dynamic, or the attribute set characterizing the objects evolves. Obviously, for obtaining in these conditions a partitioning of the object set, the clustering algorithm can be applied over and over again, beginning from the scratch, every time when the objects or attributes change. But this can be unefficient. What we want is to propose an incremental, *k-means* based clustering method, named *Core Based Incremental Clustering (CBIC)*, that is capable to efficiently re-partition the objects set, when the attributes set increases with one new attribute. The method starts from the partitioning into clusters that was established by applying *k-means* or *CBIC* before the attribute set changed. The result is reached more efficiently than running *k-means* again from the scratch on the feature-extended object set.

2. Formal Problem Study

Let $\{O_1, O_2, \ldots, O_n\}$ be the set of objects to be classified. Each object is measured with respect to a set of m initial attributes and is described therefore by a m-dimensional vector $O_i = (O_{i1}, \ldots, O_{im}), O_{ik} \in \Re, 1 \leq i \leq n, 1 \leq k \leq m$. Usually, the attributes associated to objects are standardized, in order to ensure an equal weight to all of them ([6]).

Let $\{K_1, K_2, \ldots, K_p\}$ be the set of clusters discovered in data by applying the *k-means* algorithm. Each cluster is a set of objects, $K_j = \{O_1^j, O_2^j, \ldots, O_{n_j}^j\}, 1 \leq j \leq p$. The centroid (clusters mean) of the cluster K_j is denoted by f_j, where $f_j = \left(\frac{\sum_{k=1}^{n_j} O_{k1}}{n_j}, \ldots, \frac{\sum_{k=1}^{n_j} O_{km}}{n_j} \right)$.

The measure used for discriminating objects can be any *metric* function, d. We used the *Euclidian distance*: $d(O_i, O_j) = d_E(O_i, O_j) = \sqrt{\sum_{l=1}^{m} (O_{il} - O_{jl})^2}$.

The measured set of attributes is afterwards extended with one new attribute, the $(m+1)$ or last attribute. After extension, the objects' vectors become $O_i' = (O_{i1}, \ldots, O_{im}, O_{i,m+1}), 1 \leq i \leq n$.

We want to analyze the problem of recalculating the objects grouping into clusters, after object extension and starting from the current partitioning. We want to obtain a performance gain in respect to the partitioning from scratch process.

We start from the fact that, at the end of the initial clustering process, all objects are closer to the centroid of their cluster than to any other centroid. So, for any cluster j and an object $O_i^j \in K_j$, inequality (1) holds.

$$d_E(O_i^j, f_j) \leq d_E(O_i^j, f_r), 1 \leq r \leq p, r \neq j. \tag{1}$$

We denote by $K'_j, 1 \leq j \leq p$ the set containing the same objects as K_j, after the extension. By $f'_j, 1 \leq j \leq p$ we denote the mean (center) of the set K'_j. These sets $K'_j, 1 \leq j \leq p$ will not necessarily represent clusters after the attribute-set extension. The newly arrived attribute can change the objects arrangement into clusters, formed so that the intra-cluster similarity to be high and inter-cluster similarity to be low. But there is a considerable chance, when adding one or few attributes to objects, and the attributes have equal weights and normal data distribution, that the old arrangement in clusters to be close to the actual one. The actual clusters could be obtained by applying the *k-means* classification algorithm on the set of extended objects. But we try to avoid this process and replace it with one less expensive but not less accurate. With these being said, we agree, however, to continue to refer the sets K'_j as clusters.

We therefore start by taking as reference point the previous partitioning in clusters and study in which conditions an extended object $O_i^{j\prime}$ is still correctly placed in its cluster K'_j. For that, we express the distance of $O_i^{j\prime}$ to the center of its cluster, f'_j, compared to the distance to the center f'_r of any other cluster K'_r.

Theorem 1. When inequality (2) holds for an extended object $O_i^{j\prime}$ and its cluster K'_j

$$O_{i,m+1} \geq \frac{\sum_{k=1}^{n_j} O_{k,m+1}}{n_j} \tag{2}$$

then the object $O_i^{j\prime}$ is closer to the center f'_j than to any other center $f'_r, 1 \leq r \leq p, r \neq j$.

Proof

We prove below this statement.

$$d^2(O_i^{j\prime}, f'_j) - d^2(O_i^{j\prime}, f'_r) = d^2(O_i^j, f_j) + \left(\frac{\sum_{k=1}^{n_j} O_{k,m+1}}{n_j} - O_{i,m+1}\right)^2 - d^2(O_i^j, f_r) - \left(\frac{\sum_{k=1}^{n_r} O_{k,m+1}}{n_r} - O_{i,m+1}\right)^2.$$

Using the relation in (1), we have:

$$d^2(O_i^{j'}, f'_j) - d^2(O_i^{j'}, f'_r) \leq \left(\frac{\sum_{k=1}^{n_j} O_{k,m+1}}{n_j} - O_{i,m+1} \right)^2 - \left(\frac{\sum_{k=1}^{n_r} O_{k,m+1}}{n_r} - O_{i,m+1} \right)^2 \Leftrightarrow$$

$$d^2(O_i^{j'}, f'_j) - d^2(O_i^{j'}, f'_r) \leq \left(\frac{\sum_{k=1}^{n_j} O_{k,m+1}}{n_j} - \frac{\sum_{k=1}^{n_r} O_{k,m+1}}{n_r} \right) \cdot \left(\frac{\sum_{k=1}^{n_j} O_{k,m+1}}{n_j} + \frac{\sum_{k=1}^{n_r} O_{k,m+1}}{n_r} - 2 \cdot O_{i,m+1} \right).$$

If the relation in (2) holds for $O_i^{j'}$, then the inequality above becomes:

$$d^2(O_i^{j'}, f'_j) - d^2(O_i^{j'}, f'_r) \leq - \left(\frac{\sum_{k=1}^{n_j} O_{k,m+1}}{n_j} - \frac{\sum_{k=1}^{n_r} O_{k,m+1}}{n_r} \right)^2 \Leftrightarrow$$

$$d^2(O_i^{j'}, f'_j) - d^2(O_i^{j'}, f'_r) \leq 0.$$

Because all the distances are non-negative, it results that

$$d(O_i^{j'}, f'_j) \leq d(O_i^{j'}, f'_r).$$

3. The *Core Based Incremental Clustering* Algorithm

We will use the property enounced in the previous paragraph in order to identify inside each cluster $K'_j, 1 \leq j \leq p$ those objects that have a considerable chance to remain stable in their cluster. We will use these *cluster cores* as seed for clustering.

Definition 1. We denote by $Core_j = \{O_i^{j'} | O_i^{j'} \in K'_j, d(O_i^{j'}, f'_j) \leq d(O_i^{j'}, f'_r), 1 \leq r \leq p, r \neq j\}$ the set of all objects in K'_j that are closer to f'_j than to any other center f'_r. We denote by $CORE$ the set $\{Core_j, 1 \leq j \leq p\}$ of all clusters cores.

All objects in $Core_j$ will surely remain together in the same group if clusters do not change. This will not be the case for all core objects, but for most of them.

We give next the Core Based Incremental Clustering algorithm.

We mention that the algorithm stops when the clusters from two consecutive iterations remain unchanged or the number of steps performed exceeds the maximum number of iterations allowed.

`Algorithm Core Based Incremental Clustering is`

Input: - `the set` $X = \{O_1, \ldots, O_n\}$ `of m-dimensional objects previously clustered,`
- `the set` $X' = \{O'_1, \ldots, O'_n\}$ `of (m+1)-dimensional extended objects to be clusterized,` O'_i `has the same first m components as` O_i`,`
- `the metric` d_E `between objects in a multi-dimensional space,`
- `p, the number of desired clusters,`
- $F = \{F_1, \ldots, F_p\}$ `the previous partitioning of objects in` X`.`
- $noMaxIter$ `the maximum number of iterations allowed.`

Output: - `the re-partitioning` $F' = \{F'_1, \ldots, F'_p\}$ `for the objects in` X'

Begin
For all clusters $F_j \in F$
Calculate $Core_j = \{O_i^{j'} \in F_j'$ that satisfies inequality (2)$\}$
$F_j' := Core_j$
Calculate f_j' as the mean of objects in $Core_j$
EndFor
While (F' changes between two consecutive steps) and
(there were not performed $noMaxIter$ iterations) do
For all clusters F_j' do
$F_j' := \{O_i' \mid \forall f_r'\ d(O_i', f_j') \leq d(O_i', f_r')\}$
EndFor
For all clusters F_j' do
$f_j' :=$ the mean of objects in F_j'
EndFor
EndWhile
End.

4. Results and Evaluation

In this section we present some experimental results obtained after applying the CBIC algoritm described in section 3.

For this purpose, we had used a programming interface for non-hierarchical clustering described in ([1]). We have to mention that using this interface we can simply develop non-hierarchical clustering applications for different kind of data (objects to be clusterized). As it is shown in our experiments, the objects to be clusterized are very different (patients, wine instances).

As a case study, for experimenting our theoretical results described in section 2 and for evaluating the performance of the CBIC algorithm, we consider some experiments that are briefly described in the following subsections.

We have to mention that all data were taken from the website at "http://www.cormactech.com/neunet".

As a quality measure we take the movement degree of the core objects. More stable they are, better was the decision to choose them as cores for the incremental clustering process. We express the *core stability factor* as:

$$CSF(CORE) = \frac{\sum_{j=1}^{p} \frac{|Core_j|}{no\ of\ clusters\ where\ the\ objects\ in\ Core_j\ ended}}{\sum_{j=1}^{p} |Core_j|} \tag{3}$$

The worst case is when each object in $Core_j$ ends in a different final cluster, and this happens for every core in CORE. The best case is that every $Core_j$ remains compact and it is found in a single final cluster. So, the limits between which CSF varies are given below, where the higher the value of CSF is, better was the cores choise:

$$\frac{p}{\sum_{j=1}^{p} |Core_j|} \leq CSF(CORE) \leq 1 \tag{4}$$

4.1. **Experiment 1. Cancer.** The breast cancer database was obtained from the University of Wisconsin Hospitals, Madison from Dr. William H. Wolberg.

The objects to be clusterized in this experiment are patients: each patient is identified by 9 attributes [2].The attributes have been used to represent instances. Each instance has one of 2 possible classes: benign or malignant. In this experiment are 457 patients (objects).

The attribute information used in the "cancer" experiment is shown in Table 1.

TABLE 1. Attribute information in the "cancer" experiment

	Attribute	Domain
1.	Clump Thickness	1 - 10
2.	Uniformity of Cell Size	1 - 10
3.	Uniformity of Cell Shape	1 - 10
4.	Marginal Adhesion	1 - 10
5.	Single Epithelial Cell Size	1 - 10
6.	Bare Nuclei	1 - 10
7.	Bland Chromatin	1 - 10
8.	Normal Nucleoli	1 - 10
9.	Mitoses	1 - 10

4.2. **Experiment 2. Dermatology.** The file for this experiment was obtained from the website at "http://www.corma-ctech.com/neunet".

The objects to be clusterized in this experiment are also patients: each patient is identified by 34 attributes, 33 of which are linear valued and one of them is nominal. There are 366 objects (patients).

The aim of the clustering process is to determine the type of Eryhemato-Squamous Disease [3].

The differential diagnosis of erythemato-squamous diseases is a real problem in dermatology [7]. They all share the clinical features of erythema and scaling, with

very little differences. The diseases in this group are psoriasis, seboreic dermatitis, lichen planus, pityriasis rosea, cronic dermatitis, and pityriasis rubra pilaris.

Usually a biopsy is necessary for the diagnosis but unfortunately these diseases share many histopathological features as well. Another difficulty for the differential diagnosis is that a disease may show the features of another disease at the beginning stage and may have the characteristic features at the following stages.

Patients were first evaluated clinically with 12 features. Afterwards, skin samples were taken for the evaluation of 22 histopathological features. The values of the histopathological features are determined by an analysis of the samples under a microscope.

In the dataset constructed for this domain, the family history feature has the value 1 if any of these diseases has been observed in the family, and 0 otherwise. The age feature simply represents the age of the patient. Every other feature (clinical and histopathological) was given a degree in the range of 0 to 3. Here, 0 indicates that the feature was not present, 3 indicates the largest amount possible, and 1, 2 indicate the relative intermediate values.

4.3. **Experiment 3. Wine.** The file for this experiment was obtained from the website at "http://www.corma-ctech.com/neunet".

These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines [4].

The objects to be clusterized in this experiment are wine instances: each is identified by 13 attributes. There are 178 objects (wine instances).

We have to mention that all attributes in this experiment are continuous.

4.4. **Results.** In this section we present comparatively the results obtained after applying the CBIC algorithm for the experiments described in the above subsections.

TABLE 2. The comparative results

Experiment	No. of objects	No. of attributes (m+1)	No. of iterations for m+1 attributes	No. of iterations for m attributes	No. of iterations for m+1 attributes using CBIC	The cores' stability factor CSF(CORE)
Cancer	457	9	13	10	8	0.804347826
Dermatology	366	34	7	11	5	0.713114754
Wine	178	13	4	6	3	1.0

From Table 2 we observe that using the CBIC algorithm the number of iterations for finding the solution is smaller, and also the cores' stability factor, CSF(CORE), is high.

5. Conclusions and Future Work

We proposed in this paper a new method for adapting a clustering when the attribute set describing the objects increases by one. The experiments on different data sets prove that the result is reached more efficiently using the proposed method than running *k-means* again from the scratch on the feature-extended object set.

Further works can be done in the following directions:

- to experiment the theoretical results in the case in which more attributes (that characterize the objects) are added;
- how can the theoretical results described for non-hierarchical clustering be applied/generalized for other clustering techniques.

References

[1] Şerban, G.: "A Programming Interface for Non-Hierarchical Clustering", Studia Universitatis "Babeş-Bolyai", Informatica, XLX(1), 2005, to appear.

[2] Wolberg, W., Mangasarian, O.L.: "Multisurface method of pattern separation for medical diagnosis applied to breast cytology", Proceedings of the National Academy of Sciences, U.S.A., Volume 87, December 1990, pp 9193–9196.

[3] Demiroz, G., Govenir, H. A., Ilter, N.: "Learning Differential Diagnosis of Eryhemato-Squamous Diseases using Voting Feature Intervals", Artificial Intelligence in Medicine.

[4] Aeberhard, S., Coomans, D., de Vel, O.: "THE CLASSIFICATION PERFORMANCE OF RDA" Tech. Rep. no. 92–01, 1992, Dept. of Computer Science and Dept. of Mathematics and Statistics, James Cook University of North Queensland.

[5] Jain, A., Dubes, R, "Algorithms for Clustering Data", Prentice Hall, Englewood Cliffs, New Jersey, 1998.

[6] Han, J., Kamber, M. "Data Mining: Concepts and Techniques", Morgan Kaufmann Publishers, 2001.

[7] http://www.cormactech.com/neunet, "Discover the Patterns in Your Data", CorMac Technologies Inc, Canada.

Babeş Bolyai University, Cluj Napoca,Romania
E-mail address: gabis@cs.ubbcluj.ro

Babeş Bolyai University, Cluj Napoca, Romania
E-mail address: alina@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume L, Number 1, 2005

REDESIGN BASED OPTIMIZATION FOR DISTRIBUTED DATABASES

HOREA-ADRIAN GREBLA, ANCA GOG

Abstract. The execution process of the queries in distributed databases require accurate estimations and predictions for performance characteristics. The problems of data allocation and query optimization done by means of mobile agents and evolutionary algorithms are considered. These problems still present a challenge because of the dynamic changes in number of components and architectural complexity of nowadays system topologies. The distributed system is modeled as a graph structure on which is defined a dynamic cost vector. The cost vector remains consistent, relevant, by use of mobile agents performing cost statistics and vector updates. An evolutionary technique for the re-design phase is proposed. Experimental results prove the efficiency of the proposed technique.

Keywords: Distributed Databases, Data Fragmentation, Data Allocation, Evolutionary Computation, Mobile agents

1. Introduction

Distributed databases (DDBs) have become necessity as networks expand and organizations perform geographically distributed operations. International companies store their data at different sites of a computer network, possibly in a variety of forms, ranging from flat files, to hierarchical, relational or object-oriented databases. The network itself consists of variety of transmission media, network topologies or network speeds. Design approaches for distributed databases have to consider various factors that can affect performance: CPU time, data transmission time, disk I/O operation time. Such distributed system architecture reveals some data management challenges. The system needs to be highly scalable with no critical failure points. In accordance to nowadays computing needs, the latency must not affect the performance of real-time applications. The aim is to provide uniform access to physically distributed data, no mater what the distance between the access location and places data resides. A possible approach is to represent the DDB as a graph and to perform system's management automatically by means

2000 *Mathematics Subject Classification.* 62E99, 68T99.

1998 *CR Categories and Descriptors.* C.2.4 [**Computer Systems Organization**]: Computer-Communication Networks – *Distributed systems* I.2.8 [**Artificial Intelligence**]: Problem solving, Control methods, and Search – *Heuristic methods*;

of mobile agents. An evolutionary algorithm is proposed to solve the problem of re-fragmentation and re-allocation of data.

2. Distributed Database Design Issues

Distributed database management system [8] has to ensure local applications for each computational component as well as global applications on more computational machines; it also has to provide a high-level query language with distributed query power, for distributed applications development. Must be ensured transparency levels that confer the image of a unique database. To improve the performance of global queries, data can be partitioned and spread over the system's components. A distributed database system supports data fragmentation if a relation stored within can be divided in pieces called fragments. These fragments can be stored on different sites residing on the same or different machines. The aim is to store the fragments closer to where they are more frequently used in order to achieve best performance. The partitions can be created horizontal, vertical or mixed (the combination of horizontal and vertical fragmentation).

Let $R[A_1,A_2,\ldots,A_n]$ be a relation where A_i, $i = 1,\ldots,n$ are attributes. A horizontal fragment can be obtained by applying a restriction: $R_i = \sigma_{cond_i}(R)$, where $cond_i$ is the guard condition. So we can rebuild the original relation by union as follows:

$$R = R_1 \cup R_2 \cup \ldots \cup R_k.$$

A vertical fragment is obtained by a projection operation:

$$R_i = \prod_{\{A_{x1},A_{x2},\ldots,A_{xp}\}}(R),$$

where A_{xi}, $i = 1,\ldots,p$ are attributes. The initial relation can be reconstructed by join of the fragments:

$$R = R_1 \bigotimes R_2 \bigotimes \ldots \bigotimes R_l.$$

A DDB system can be represented [5] as a graph where the sites are given by (V), the set of vertices, and the edges (E) given by the direct connections between sites. Each edge has associated a cost, but this cost will be examined later in this paper. For exemplification we consider in Figure 1 a distributed system and obtain in Figure 2 the corresponding graph representation.

The system must preserve distributed data independence [9], such that any change of physical location of data must not disturb application functionality. A good management of DDBs implies a considerable effort in the design phase of the system and also implies a redesign phase for performance tuning. One of the design phase component that raise problems represent data fragmentation and allocation. The biggest improvement in system's response can be achieved by fragmentation and reallocation in the design refinement phase. The use of mobile agents can bring great performance value to the system because a software agent [10] can act autonomously on behalf of the administrator. The elements of the system do not

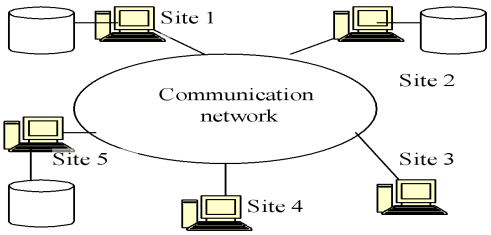


FIGURE 1. Distributed database system

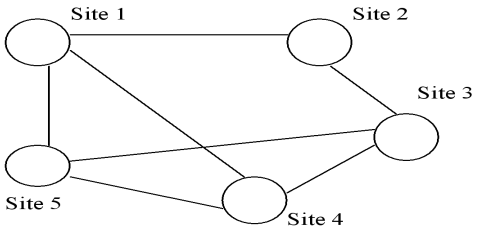


FIGURE 2. Corresponding graph of the system from Figure 1

have to be connected all the time, agents can travel in the network and execute at different hosts by taking their state and implementation with them. Agents can be intelligent, take decisions and react to environment changes to perform their actions, and most important, they can cooperate to fulfill their common goal.

3. Overview and Architecture of the System

In what follows, a distributed database system architecture where design relies on the graph representation and system management improvement by use of agents is proposed. An agent based architecture with distributed access and concurrent queries in heterogeneous database system is described. The considered architecture provides high scalability and performance optimization. The main improvement is the manner of cost definition between sites:

• First, we define the initial cost assigned by the system designer to an edge; this cost is estimated based on network transfer rate, data access time and computing power on a site. We call this ***initial estimated cost***.

• At some given times we can obtain more accurate cost in the system; we define this cost the ***up-to-date computed cost***.

Translator agents perform the translation of local names to global names and provide a common language for distributed queries assuring local database management system independence.

Retriever agents collect data from corresponding fragments by communicating with Translators. In fact they build the query in the agent common language and ask the translators for results.

Optimizer can be unique for the entire system or can be cloned; it contains the query optimizer. One role of the *Optimizer* is to build *up-to-date computed cost* from statistics for the sites with respect to the amount of data accessed on that site. The proposed architecture is depicted in Figure 3.

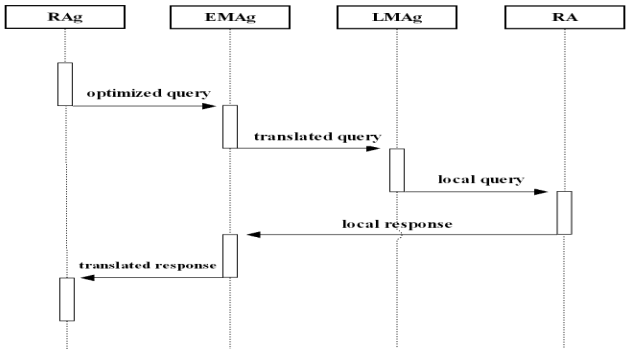


FIGURE 3. Proposed architecture

4. Evolutionary Fragmentation and Allocation Algorithm in Distributed Databases

The problem of database fragmentation and data allocation is modeled as a graph. We have to distribute m tuples to n nodes of the graph. The costs of the edges between the vertexes of the graph are given. Also, statistics referring to the frequency of the requested tuples in the graph are given (computed by agents). The tuples' distribution can be reduced to an optimization problem which goal is to minimize the costs generated by the queries in the graph. An evolutionary algorithm is proposed to solve this NP-Complete problem [6]. The proposed algorithm is called Evolutionary Fragmentation and Allocation Algorithm in Distributed Databases (EFA algorithm).

A fixed size population is used in the proposed algorithm. The m tuples that have to be distributed to nodes will be denoted by $t_1, t_2, \ldots, t_m$. There are no restrictions regarding the minimum or the maximum number of tuples contained by a node. A potential solution of the problem (a chromosome) is a string of

constant length $\{x_1, x_2, \ldots, x_m\}$, where the gene x_i, $x_i \in \{1, 2, \ldots, n\}$, indicates to which node the tuple i belongs.

The potential solutions are evaluated by means of a real-valued fitness function $F, F\colon X \to \mathbb{R}$, where X denotes the space of potential solutions. The fitness of a chromosome takes into account the costs of the edges between nodes and the statistics regarding the frequency of the requested tuples in the graph:

$$F(x) = \sum_{i=1}^{n} \sum_{j=1}^{m} f_{ij} c_{iN_j},$$

where f_{ij}, $i \in \{1, 2, \ldots, n\}$, $j \in \{1, 2, \ldots, m\}$, represents the frequency of the requests of the tuple j from the node i of the graph. Also, c_{iN_j}, $i \in \{1, 2, \ldots, n\}$, $j \in \{1, 2, \ldots, m\}$, represents the cost of the edges between the node i and the node that contains the tuple j, denoted by N_j. The fitness function F is to be minimized.

Rank-based selection for recombination mechanism [4], *two points crossover* and weak mutation operator [1] are considered for the proposed algorithm [3]. The best from parent and offspring enters the new generation [2].

The algorithm ends after a certain number of generations that did not improve the best solution of the generation [7]. The best solution obtained during the search process is considered to be the solution of the problem.

5. Experimental results

A graph having five nodes is considered ($n = 5$). Let us denote the five nodes by N_1, N_2, N_3, N_4, N_5. The associated costs for the edges between the given nodes are depicted in Figure 4. *Remark:* We are interested only in the direct cost

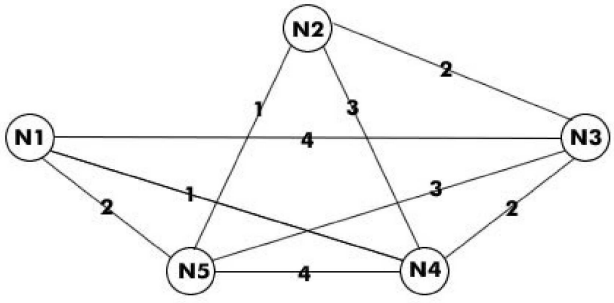


Figure 4. Proposed architecture

between two nodes, and that is why there are nodes without edges between them, even if there could be a path between the two nodes by using intermediate nodes.

A dataset of 1.200.000 tuples is given. The given tuples are denoted by $t_1, t_2, \ldots, t_m$, where m represents the number of tuples. The existing dataset fragmentation and distribution of tuples in nodes are depicted in the Table 1.

TABLE 1. Dataset fragmentation and distribution of tuples in nodes.

Node	**Dataset fragmentation**	**Number of tuples/node**
N_1	$t_1 - t_{100.000}$	100.000
N_2	$t_{100.001} - t_{380.000}$	280.000
N_3	$t_{380.001} - t_{540.000}$	160.000
N_4	$t_{540.001} - t_{720.000}$	180.000
N_5	$t_{720.001} - t_{1.200.000}$	480.000

The statistics regarding the frequency of requests of the tuples from each node are depicted in Tables 2 - 6.

TABLE 2. The frequency of requests of the tuples from the node N_1

Tuples	**Frequency**
$t_{40.001} - t_{90.000}$	4
$t_{420.001} - t_{560.000}$	10
$t_{610.001} - t_{730.000}$	12
$t_{980.001} - t_{1.100.000}$	5

TABLE 3. The frequency of requests of the tuples from the node N_2

Tuples	**Frequency**
$t_{250.001} - t_{330.000}$	2
$t_{560.001} - t_{680.000}$	14
$t_{1.100.001} - t_{1.200.000}$	7

TABLE 4. The frequency of requests of the tuples from the node N_3

Tuples	**Frequency**
$t_1 - t_{100.000}$	3
$t_{250.001} - t_{290.000}$	10
$t_{880.001} - t_{970.000}$	9
$t_{990.001} - t_{1.000.000}$	16

The tuples that do not appear in the tables containing the frequency of requests are never requested. They will remain inside the nodes that contain them before applying the EFA algorithm. The proposed EFA algorithm was applied for data described above. The chosen values for the algorithm parameters are written in Table 7.

TABLE 5. The frequency of requests of the tuples from the node N_4

Tuples	**Frequency**
$t_{100.001} - t_{170.000}$	3
$t_{220.001} - t_{330.000}$	7
$t_{450.001} - t_{560.000}$	13
$t_{680.001} - t_{770.000}$	8

TABLE 6. The frequency of requests of the tuples from the node N_5

Tuples	**Frequency**
$t_{200.001} - t_{260.000}$	12
$t_{700.001} - t_{830.000}$	6
$t_{920.001} - t_{980.000}$	1

TABLE 7. The EFA algorithm parameters

Population size	**Number of generations that did not improve the current solution**	**Probability of recombination**	**Probability of mutation**
200	50	0.7	0.1

After applying EFA algorithm, the way the tuples are redistributed to the nodes of the graph, by taking into account the frequency of the requests of the tuples, is described in Table 8.

6. CONCLUSIONS AND FUTURE WORK

An evolutionary algorithm called EFA was proposed for the redesign phase, meaning re-fragmentation and re-allocation, in our distributed system. The considered problem is a NP-Complete one. EFA was successfully applied and experimental results have proved the efficiency of the proposed algorithm.

As future work, the method can be improved by computing the costs weighted with factors like local interest for fragments (recommend replication or not), real-time response importance (some applications do not need real-time response), data access frequency (balance sheet data may be consulted once in a month).

The weight of the factors in the cost computation can be changed in time, also changes in network topology or transmission media can influence the response time. The statistics are useful for rebalancing the system by re-computing the costs to obtain best response time for all queries on any site.

TABLE 8. Reallocation of tuples in nodes after applying EFA.

Node	Dataset refragmentation	Number of tuples/node
N_1	$t_{680.001} - t_{770.000}$, $t_{1.000.001} - t_{1.100.000}$	190.000
N_2	$t_{170.001} - t_{200.000}$, $t_{250.001} - t_{260.000}$ $t_{330.001} - t_{380.000}$, $t_{560.001} - t_{610.000}$ $t_{770.001} - t_{830.000}$, $t_{970.001} - t_{980.000}$ $t_{1.100.001} - t_{1.200.000}$	310.000
N_3	$t_1 - t_{40.000}$, $t_{90.000} - t_{100.000}$ $t_{260.001} - t_{330.000}$, $t_{380.001} - t_{420.000}$ $t_{880.001} - t_{970.000}$, $t_{990.001} - t_{1.000.000}$	260.000
N_4	$t_{40.001} - t_{90.000}$, $t_{100.001} - t_{170.000}$ $t_{420.001} - t_{560.000}$, $t_{980.001} - t_{990.000}$	270.000
N_5	$t_{200.001} - t_{250.000}$, $t_{610.001} - t_{680.000}$ $t_{830.001} - t_{880.000}$	170.000

REFERENCES

[1] Bäck, T., Fogel, D.B., Michalewicz, Z. (Editors), *Handbook of Evolutionary Computation*, Institute of Physics Publishing, Bristol and Oxford University Press, New York, 1997.

[2] Bäck, T., *Optimal mutation rates in genetic search*, Proceedings of the 5th International Conference On Genetic Algorithms, Ed. S. Forrest, Morgan Kaufmann, San Mateo, CA, 2-8, 1993.

[3] Dumitrescu, D., Lazzerini, B., Jain, L.C, Dumitrescu, A., *Evolutionary Computation*, CRC Press, Boca Raton, FL., 2000.

[4] Goldberg, D.E., Deb, K., *A comparative analysis of selection schemes used in genetic algorithms*, Foundations of Genetic Algorithms G.J.E. Rawlins (Ed.), Morgan Kaufmann, San Mateo, CA, 69-93, 1991.

[5] Moldovan, G., *Reorganization of a Distributed Database*, Babes-Bolyai University, Seminar of Models, Structures and Information Processing, Preprint nr. 5, p. 3-10, 1984.

[6] Levin, K. D., Morgan, H. L., *Optimizing distributed databases-A framework for research*, Proceedings of AFZPS NCC, vol. 44. AFIPS Press, pp. 473-478, 1975.

[7] Mitchell, M., *An Introduction to Genetic Algorithms*, MIT Press, Cambridge, MA, 1996.

[8] Oszu, M. T., Valduriez, P., *Principles of Distributed Database Systems*, Prentice Hall, Englewood Cliffs, NJ, 1999.

[9] Piattini, M. and Diaz, O., *Advanced Database Technology and Design*, Artech House, Inc. 685 Canton Street Norwood, MA 02062, 2000.

[10] Weiss, G., *Multiagent System, A Modern Approach to Distributed Artificial Intelligence*, MIT Press , USA, 2000.

BABES-BOLYAI UNIVERSITY OF CLUJ-NAPOCA, FACULTY OF MATHEMATICS AND COMPUTER SCIENCE, COMPUTER SCIENCE DEPARTMENT

E-mail address: horea,anca@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume L, Number 1, 2005

PARALLEL LAGRANGE INTERPOLATION ON EXTENDED FIBONACCI CUBES

IOANA ZELINA

Abstract. In this paper is presented a parallel algorithm for computing a Lagrange interpolation on a Extended Fibonacci Cube $EFC_1(n)$.The algorithm consists of three phases: initialisation phase, main phase in wich the Lagrange polynomials are computed and final phase in wich the terms of the interpolation formula are added together.

1. Introduction

Interpolation techniques are of great importance in numerical analysis since they are used in many science and engineering domains. The Lagrange interpolation for a given set of points $(x_1, y_1), (x_2, y_2)$,..., (x_N, y_N) and a value x is defined as

$$f(x) = \sum_{i=1}^{N} y_i \times L_i(x) \tag{1.1}$$

where $L_i, i = \overline{1, N}$ are the Lagrange polynomials given by the formula

$$L_i(x) = \frac{(x - x_0) \dots (x - x_{i-1})(x - x_{i+1}) \dots (x - x_N)}{(x_i - x_0) \dots (x_i - x_{i-1})(x_i - x_{i+1}) \dots (x_i - x_N)} \tag{1.2}$$

When the number of points N is very large a long computation time and a large storage capacity may be required to carry out the calculation. To overcome this, a parallel implementation would be appropriate. This kind of parallel algorithms were introduced for Lagrange interpolation for different topologies: Goertzel [2] has introduced a parallel algorithm for a tree topology with N processors augmented with ring connections which requires $N/2 + O(\log N)$ steps each composed of two substractions and four multiplications; a parallel algorithm has been discussed in [6] which uses a k-ary n-cube consisting of $O(k^n + kn)$ steps, each with 4 multiplications and substractions for $N = k^n$ node interpolation. In [5] is described a parallel algorithm for computing a $N = n!$-node Lagrange interpolation on a n-star graph. The algorithm in [5] consists of three phases and requires $n!/2$ steps, each consisting of 4 multiplications, 4 substractions and one communication operation. In [7] this parallel algorithm is applied for computing an $N = n2^n$ point Lagrange interpolation on an n-dimensional cube-connected cycles (CCC_n).

Received by the editors: February 18, 2005.

The method can be applied for any Hamiltonian network, the performances depending on the communication abilities of the host network.

In this paper the algorithm described in [5] is applied to a Extended Fibonacci Cube topology. The algorithm relies on all-to-all broadcast communication at some stages during computation. This is achieved by using a gossiping algorithm on a ring embedded in the host network having its all nodes.

2. Preliminaries

Due to the regularity, logarithmic diameter, logarithmic node degrees, etc. of hypercube interconnection networks, they are used by most researchers. The hypercube provides a rich interconnection structure which permits many other topology to be emulated. Nevertheless, when dimension of hypercube increases, the number of nodes increases too fast. Because of this, Hsu [3] developed Fibonacci Cubes and they are more sparse than the hypercubes and Wu [8] developed Extended Fibonacci Cubes by changing the initial conditions. The Extended Fibonacci Cubes are also more sparse than the hypercubes.

One third of Fibonacci Cubes are Hamiltonian, however, all of Extended Fibonacci Cubes are Hamiltonian. Both of these interconnection can be considered as nodes faulty with incident edges hypercubes.

Fibonacci Cubes and Extended Fibonacci Cubes

Extended Fibonacci Cubes (EFC) topology was proposed by Wu [1] and this topology is based on the Fibonacci Cube proposed by Hsu [3]. Both topologies use the Fibonacci series and initial conditions for topologies can be different from Fibonacci series initial conditions.

An k^{th} $(k = 1, 2)$ order Extended Fibonacci Cube is denoted by $EFC_k(n)$ where $n-2$ is the length of bitstring representing the address of nodes in EFC. $EFC_k(n)$ is a subgraph of the corresponding hypercube. Each node of $EFC_k(n)$ is addressed with Fibonacci Code (FC). The simplest version of these cubes series is the Fibonacci Cubes. The Fibonacci Cube ($FC(n)$) can be described as below.

Definition 2.1. [Fibonacci Cube] *Assume the graphs $FC(n) = (V(n), E(n))$, $FC(n-1) = (V(n-1), E(n-1))$ and $FC(n-2) = (V(n-2), E(n-2))$. We define the Fibonacci Cube using the recursion for the nodes set as $V(n) = 0\|V(n-1) \cup 10\|V(n-2)$, where $\|$ denotes the concatenation of two bit-strings. Two nodes in $FC(n)$ are connected by an edge in $E(n)$ if and only if their labels differ exactly in 1-bit position. The initial condition for recursion is $V(2) = \emptyset$ and $V(3) = \{0, 1\}$.*

Definition 2.2. [Extended Fibonacci Cube] *Let $EFC_1(n) = (V_1(n), E_1(n))$, where $V_1(n)$ is the set of nodes and $E_1(n)$ is the set of edges in $EFC_1(n)$, and*

$$EFC_1(n-1) = (V_1(n-1), E_1(n-1)), EFC_1(n-2) = (V_1(n-2), E_1(n-2)).$$

$EFC_1(n)$ can be defined recursively by using $EFC_1(n-1)$ and $EFC_1(n-2)$ as it follows: $V_1(n) = 0\|V_1(n-1) \cup 10\|V_1(n-2)$ where $\|$ denotes the concatenation of two strings. The initial condition for recursion is $V_1(3) = \{0, 1\}$ and $V_1(4) = \{00, 10, 11, 01\}$. Two nodes in $EFC_1(n)$ are connected if and only if their address representations differ in exactly 1-bit position.

Some $EFC_1(n)$ are shown in Fig.1 where $n = 3, 4, 5, 6$ and each $EFC_1(n)$ consists of an $EFC_1(n-1)$ and an $EFC_1(n-2)$.

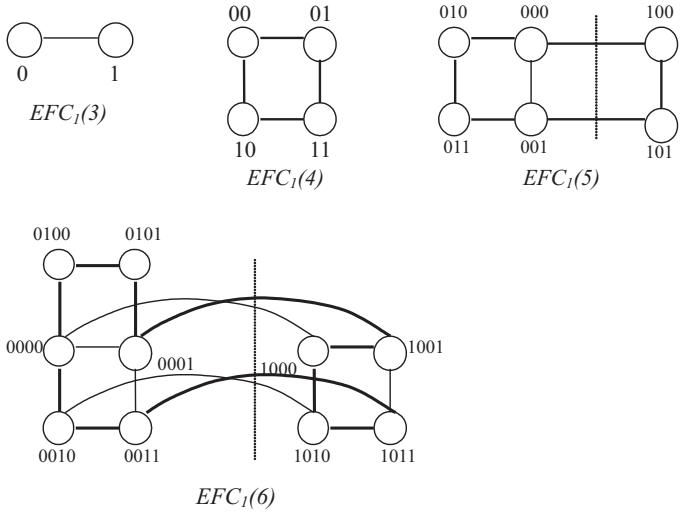


Fig. 1

It is known that while less than one third of Fibonacci cubes are hamiltonian, all of $EFC_k(n)$ are Hamiltonian. This can be proved using inductive reasoning on n, where $n = 4$ and $n = 5$ are the induction basis. In Fig. 1 a hamiltonian cycle is shown with bold links. This means that ring can be embedded into $EFC_k(n)$ with dilation and congestion 1. $FC(n)$ is a proper subgraph of $EFC_1(n)$.

There is a Hamming distance path in $EFC_1(n)$ where Hamming distance is the exclusive-or operation on both addresses of nodes and this distance is equal to Hamming distance. The diameter of $EFC_1(n)$ is $n-2$ and node degrees are between $\lceil \frac{n}{3} \rceil$ and $n-2$.

By changing the initial conditions for $EFC_1(n)$, another Extended Fibonnaci Cube can be extracted, denoted as $EFC_2(n)$ with initial conditions $V_2(4) = \{00, 10, 11, 01\}$ and $V_2(5) = \{000, 100, 101, 111, 110, 010, 011, 001\}$.

$EFC_1(n)$ is a proper subgraph of $EFC_2(n)$. The generated FC for $EFC_1(n)$ and $EFC_2(n)$ are mutually disjoint.

We denote by N the number of nodes in $EFC_1(n)$.

3. The Parallel Algorithm

The parallel algorithm is based on the algorithm described in [5] for computing a $N = n!$ node Lagrange interpolation on a n-star graph. We shall apply this algorithm for a network using an n- extended Fibonacci cube $EFC_1(n)$ topology with bidirectional links between nodes. Let N be the number of the nodes in $EFC_1(n)$.

The computation is carried out in three phases: initialisation, main and final phase. In the initialisation phase, the set of points to be interpolated are allocated to the nodes, one point for each node. Then, in the main phase, the Lagrange polynomials $L_i(x)$,$i = \overline{1,N}$ are computed and in the final phase the terms are added together to obtain the final result $y = f(x)$.

We denote by P_w the processor in the node of the extended Fibonacci cube $EFC_1(n)$ with the binary representation w. Each processor P_w has six registers denoted R_1, R_2, R_3, R_4, R_5, R_6 and we indicate by $P_w(R_i)$ the content of the register R_i in the processor P_w, $i = \overline{1,6}$, $w \in V_1(n)$ and by $P_w^{(t)}(R_i)$, $i = \overline{1,6}$, $w \in V_1(n)$ the content of the register R_i in the processor P_w after step t. In each node, registers R_1, R_2, R_3, R_4 will hold the terms required for computing the polynomials and registers R_5, R_6 will be used to implement an all-to-all broadcast algorithm in a ring embedded in the host network $EFC_1(n)$ during the main phase.

The $EFC_1(n) = (V(n), E(n))$ is hamiltonian. When constructing a hamiltonian cycle in $EFC_1(n)$, two arrays, $Next[w]$ and $Previous[w]$, which indicate the nodes before, respectively after node w, $w \in V(n)$ in the embedded cycle can also be constructed. For any node P_w in the embedded hamiltonian ring, the next and previous nodes are $P_{Next[w]}$, respectively $P_{Previous[w]}$. Those arrays should have been set to their proper values before starting the initialisation phase.

3.1. **Initialisation phase.** The values x, $Next[w]$, $Previous[w]$, (x_i, y_i) are assignated to the processor P_w to be stored in the local memory where i is the order of the Fibonacci number $w = f_i$, $i = \overline{1,N}$ with initial conditions from Definition 2.2. The registers R_1, R_2, R_3, R_4, R_5, R_6 of each processor are set to their initial values, for all $w \in V(n)$, in parallel:

$$P_w^{(0)}(R_1) = 1; \quad P_w^{(0)}(R_2) = 1; \quad P_w^{(0)}(R_3) = x_i; \quad P_w^{(0)}(R_1) = x_i;$$
$$P_w^{(0)}(R_5) = x - x_i; \quad P_w^{(0)}(R_6) = x - x_i;$$

3.2. **Main phase.** In this phase, each node P_w uses the values $Next[w]$ and $Previous[w]$ to communicate with the next and previous node in the embedded hamiltonian cycle. To compute the terms $L_i(x)$ all the processors perform the following sequence simultaneously:

For $t = 0, 1, ..., N/2 - 2$ do

$$P_w^{(t+1)}(R_3) \Leftarrow P_{Next[w]}^{(t)}(R_3);$$
$$P_w^{(t+1)}(R_4) \Leftarrow P_{\mathrm{Pr}evious[w]}^{(t)}(R_4);$$
$$P_w^{(t+1)}(R_5) \Leftarrow P_{Next[w]}^{(t)}(R_5);$$
$$P_w^{(t+1)}(R_6) \Leftarrow P_{\mathrm{Pr}evious[w]}^{(t)}(R_6);$$
$$P_w^{(t+1)}(R_1) = P_w^{(t+1)}(R_1) \times P_w^{(t+1)}(R_5) \times P_w^{(t+1)}(R_6);$$
$$P_w^{(t+1)}(R_2) = P_w^{(t+1)}(R_2) \times (x_i - P_w^{(t+1)}(R_3)) \times (x_i - P_w^{(t+1)}(R_4));$$

end for;

$$P_w^{(N/2)}(R_3) \Leftarrow P_{Next[w]}^{(N/2-1)}(R_3);$$
$$P_w^{(N/2)}(R_1) = P_w^{(N/2)}(R_1) \times P_w^{(N/2)}(R_5);$$
$$P_w^{(N/2)}(R_2) = P_w^{(N/2)}(R_2) \times (x_i - P_w^{(N/2)}(R_3));$$

The last iteration is used to avoid multiplying the terms $(x - x_{N/2})$ and $(x_i - x_{N/2})$ twice.

Each step consists of two data communications (the first two respectively the last two communications can be realized in parallel because of bidirectional links between nodes), 2 substractions and 4 multiplications.

To conclude the main phase, all the processors execute the instruction

$$P_w^{(N/2+1)}(R_1) = \frac{P_w^{(N/2)}(R_1)}{P_w^{(N/2)}(R_2)} \times y_i.$$

Therefore, at the end of this phase $P_w(R_1) = L_i(x) \times y_i$.

In the main phase, each processor performs N data communications, $2N - 1$ multiplications, $N - 1$ substractions and one division.

3.3. **The Final Phase.** In this phase, the contents of register R_1 in all nodes are added together to obtain the final result. We can use for this a gossiping method for a ring similar to the one used in the main phase but we can also use a method similar to the addition of the content of the processors in a hypercube network topology.

Remark that if a node $w \in V(n)$ in $EFC_1(n)$ is labeled with a bit string having 1 on his first position, $w = 1u_2u_3 \dots u_{n-3}$ then $u_2 = 0$ and the node $w' = u_3 \dots u_{n-3} \in V(n-2)$ is a node in $EFC_1(n-2)$. A node $w \in V(n)$ in $EFC_1(n)$ is labeled with a bit string having 0 on his first position, $w = 0u_2u_3 \dots u_{n-3}$ then the node $w' = u_2u_3 \dots u_{n-3} \in V(n-1)$ is a node in $EFC_1(n-1)$. If we add simultaneously the content of registers R_1 in all the nodes $w = 1u_2u_3 \dots u_{n-3}$ to the content of registers R_1 in the corresponding nodes $v = 0u_2u_3 \dots u_{n-3}$ then we reduce the size of the EFC from n to $n-1$. So we can add together the partial results accumulated in registers R_1 of all nodes in n steps, each consisting of one addition and two communication operations. Each step reduces the size of problem by one until the last step, $(n-2)^{th}$ step, which complete the computation having stored the final result in register R_1 of processor $P_{0\dots0}$.

For $i = 1, 2, ..., n-2$ do
  For all $w = 0^{i-1}1u_{i+1} \dots u_{n-2}$ do in parallel

$$P_{0^iu_{i+1}\dots u_{n-2}}(R_1) = P_{0^{i-1}1u_{i+1}\dots u_{n-2}}(R_1) + P_{0^iu_{i+1}\dots u_{n-2}}(R_1);$$

  end for;

End for;

This phase includes $n-2$ additions and $n-2$ communication operations.

Let n be the order of the Extended Fibonacci Cube $EFC_1(n)$ and N be the dimension of the $EFC_1(n)$, i.e. the extended Fibonacci number defined by the recursion $f_n = f_{n-1} + f_{n-2}$, $n \geq 2$, $f_0 = 2$, $f_1 = 4$.

Theorem 3.1. *The parallel algorithm presented carries out the computation of a N-point Lagrange interpolation on a Extended Fibonacci Cube $EFC_1(n)$ in a total time of $O(N)$.*

Proof: The algorithm computes a N-point interpolation, in three phases, requiring in total $N+n-2$ data communications, $n-2$ additions, $2N-1$ multiplications, $N-1$ substractions and one division.

Remark 3.1. *The parallel algorithm carries out in a total time of $O(N)$ while the running time for such an interpolation on a single-processor system is of $O(N^2)$.*

References

[1] Akl, S., *Parallel Computation: Models and Methods*, Prentice Hall, 1997

[2] Goertzel, B., *Lagrange interpolation on a tree of processors with ring connections*, JPDC, 22, pp.321-333, 1994

[3] Hsu, W.J., *Fibonacci Cube- A New Interconnection Topology*, IEEE Trans. on Parallel and Distributed Systems, vol.4, no.1, pp.3-12, 1993

[4] Karci, A., *New Interconnection Networks: Fibonacci Cube and Extended Fibonacci Cubes Based Hierarchic Networks*, Proc. of 15th ICOIN, 2001

[5] Sarbazi-Azad, H., Ould-Khaoma, M., Mackenzie, L.M., *A Parallel Lagrange Interpolation on the Star Graph*, Proc. 14th IPDPS, Cancun, Mexico, pp.777, 2000

[6] Sarbazi-Azad, H., Ould-Khaoma, M., Mackenzie, L.M., *A Parallel Lagrange Interpolation on k-ary n-cubes*, LNCS1557, pp.85-95, 1999

[7] Sarbazi-Azad, H., Ould-Khaoma, M., Mackenzie, L.M., *An Efficient Parallel Algorithm for Lagrange Interpolation and Its Performance*, Proc. 4th Int.Conf. on High Performance Conputing in Asia Pacific Region, vol 2, Beijing, China, pp.593, 2000

[8] Wu, J., *Extended Fibonacci Cubes*, IEEE Trans. on Parallel and Distributed Systems, vol.8, no.12, pp.1203-1210, 1997

North University of Baia Mare, Faculty of Science, Department of Mathematics and Computer Science, Victoriei 76, Baia Mare, Romania

E-mail address: ioanazelina@yahoo.com

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume L, Number 1, 2005

MATHEMATICS AND COMPUTER SCIENCE III – ALGORITHMS, TREES, COMBINATORICS AND PROBABILITIES, MICHAEL DRMOTA, PHILIPPE FLAJOLET, DANIÈLE GARDY AND BERNHARD GITTENBERGER (EDITORS), TRENDS IN MATHEMATICS, BIRKHÄUSER VERLAG, BASEL-BOSTON-BERLIN 2004, XV + 555 PP, ISBN: 3-7643-7128-5

RADU LUPŞA

The book contains invited papers, contributed papers (lectures) and short communications (posters), which were presented at the International Colloquium of Mathematics and Computers Science held at the Vienna University of Technology, in September 13-17, 2004. This colloquium is the third one in a now regularly established series, following the first two venues in September 2000 and September 2002 in Versailles. Their Proceedings were published too with Birkhäuser Verlag in 2000 and 2002, respectively. These colloquia were acknowledged as a success by the two communities, mathematicians and computer scientists, as well as other people working in various areas of applied mathematics and engineering. They offer the opportunity to establish the state of the art and, at a same time, to present new results, new trends and new ideas in common areas.

The present volume addresses problems situated at the interface between mathematics and Computer Science, with special emphasis on discrete probabilistic models and their relation to algorithms. Combinatorial and probabilistic properties of random graphs, random trees, combinatorial stochastic processes (random walks, for instance) are also included. The major field of applications is the analysis of algorithms and data structures, but applications to statistical theory, information theory and mathematical logic are also considered.

The papers are grouped in seven parts: I. *Combinatorial and Random Structures* (8 papers dealing with partitions, iterated logarithm law for random permutation); II. *Graph Theory* (5 papers on perfect matching in random graphs, avalanche polynomials, spanning trees in graphs, etc); III. *Analysis of Algorithms* (7 papers on move-to-rule rule with random weights, probabilistic bin packing, universal data compression, etc); IV. *Trees* (9 papers on multidimensional interval trees, monotonically labelled trees,number of vertices in a Galton-Watson forest); V.

Received by the editors: March 9, 2005.

Probability (10 papers on geometrically distributed samples, semi-Markov walks, Yaglom type limit theorems, and others); VI. *Combinatorial Stochastic Processes* (6 papers on jumping particles, Euler orientations of planar graphs, random walks on groups); VII. *Applications* (9 papers dealing among others with zero-one law for first-order logic on random images, stochastic chemical kinetics, decidability of simple brick codes).

Bringing together contributions in these closely related areas — mathematics and computer science, the present volume and the previous two ones, serve as an outstanding tool of information for a large audience including researchers, teachers, graduate students and engineers interested in applied mathematics, discrete mathematics and computer science. The volume emphasizes the interplay between mathematics and computer science, and the key roles each of them plays in the development of the other one.Also the range of applications is very wide and reaches beyond computer science.

Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania

E-mail address: rlupsa@cs.ubbcluj.ro