STUDIA
UNIVERSITATIS BABEŞ-BOLYAI

5 II

MATHEMATICA

3

1995

CLUJ-NAPOCA

Anul XL **1995**

STUDIA
UNIVERSITATIS BABEŞ-BOLYAI

MATHEMATICA

3

Redacţia : 3400 CLUJ-NAPOCA str. M. Kogălniceanu nr.1 ▸ Telefon : 194315

SUMAR-CONTENTS-SOMMAIRE

STUDIA UNIV. BABEŞ-BOLYAI, MATHEMATICA, XL, 3, 1995

SOFTWARE QUALITY MANAGEMENT: TO UNDERSTAND AND TO INTRODUCE

I.

Ecaterina BALLA[*]

Received: 15.12.1994
AMS subject classification: 68N05

REZUMAT. Controlul calităţii produselor program (I). După o scurtă trecere în revistă a definiţiilor de calitate a programelor, se prezintă cel mai popular standard de calitate: ISO 9000. Sunt discutate avantajele şi dezavantajele lui. Soluţiile schiţate, bazate pe studiul şi experienţa autoarei, pot fi aplicate în toate ţările din Europa de Est, problemele de rezolvat fiind similare.

ABSTRACT. - Following a brief exposition of the history of software-quality definitions, today's most popular approach to the problem is presented: building quality management systems according to ISO 9000 Series. Some advantages and disadvantages of this trend are presented. Using results of software quality oriented research and practical training done by the author in Hungary, some problems are shown and some solutions are sketched to start introducing software quality management. The problems sketched are similar in most Eastern European countries, therefore we presume that presented solutions could be applicable as well.

Acknowledgment. Information contained in this paper is result of research and practical training supported by IQSOFT.

1. Introduction. The term quality, as "degree of excellence, relative nature or kind of character, class or grade of thing as determined by this, general excellence" has, both in philosophy and in ordinary speech, a long history. The meaning of quality has changed over

[*] *Technical University, Department of Mathematics and Computer Science, Budapest, Hungary, E-mail: balla@inf.bme.hu*

time according to the mentality of people, but "good quality" has always had something to do with the product's capability to satisfy the needs of the users. (Product- for some people meaning a thing, for others meaning a result of mental activity.) Requirements for quality have changed as well. For centuries quality has been mostly "a matter of conscience" and seldom an expressed need of some, wealthy groups. However, quality requirements have evolved, and this evolution has accelerated mainly in our century. The first product responsibility trial was registered in 1852 in New York (purchaser's rights had to be defended versus a company using improper inscription on chemical substances). Beginning with the 1950's many quality organizations and bodies have shown up, some of them becoming of national or international importance. Due to the work of these organizations, standards and prescriptions regarding quality have been issued, and nowadays market-position of companies highly depend on the application of these standards.

According to Armand Feigenbaum [Feigen93] world-wide market-competition has led to a change in purchasers' value judgment: they calculate in life-cycle costs, so quality comes into prominence regarded to price.

2. Quality of software. Software is "intellectual creation comprising the programs, procedures, rules and any associated documentation pertaining to the operation of a data processing system", a software product is the "complete set of computer programs, procedures and associated documentation and data designated for delivery to a user" [ISO 9000-3].
The concept of software as used today exists since the NATO conference organized in 1968 in Garmisch Partenkirchen, so it is far from being a novelty. Due to the increasing number of software users, quality is now of the same importance in developing, using and maintaining of software as in the similar activities regarding any other - material or intellectual - products. Quality of software is a concept hard to define, basically because the difficulties of finding adequate quality-criteria, attributes and proper quality-measuring methods and tools.

2.1. Basic orientations in defining the meaning of "software quality". In the "heroic age" of computer science the programs were written in most cases by mathematicians, by physicists or by engineers, and were usually concerned with the solving of some scientific problems (numeric approach of solutions of equations, inverting a matrix etc.). These programs were dealing with simple mathematical operations that had to be performed many times. *A program was considered to be "good" if it was able to run - at least once, to terminate after a time - the length of running-time was not really important, and to produce results similar to those expected.*

In the sixties and the early seventies - mainly due to the appearance of assembly and high level programming languages and operating systems - the dimensions of the problems to be solved have increased, and there was a need for programs to perform the same operations thousands of times a day. It was the age of "tricky" programmers, quality being characterized by the so-called *"micro-efficiency". A program was said to be "good" if its use was cheap in a certain hardware-software environment, and it handled the given resources of the computer in an optimal way.*

Beginning with the eighties the concept of quality has started to mean *"macro-efficiency"*. Programs are being used in various hardware-software environments by many people among whom the percentage of "ordinary users" (not informaticians) is increasing. Due to the development of hardware there is in fact no point in "saving resources". *A "good" computer program is* therefore *portable, its code can be understood by more people, and it is highly user-friendly.*

Nowadays complex program-systems are being developed. The question isn't the correctness of a certain program any more, but the correctness, reliability, integrity, interoperability of the whole system. *A system is good if it's documentation is complete and understandable, if it interacts with other - computer aided or classic - data-processing systems in an optimal way, if it's maintenance can be done at low costs, if it is able to cooperate with new software-hardware tools* and so on.

Prescriptions for coding aren't enough any more: *complex organizational frame, well-organized team-work are needed* to insure the correct and optimal sequence of all activities performed in the development of such systems.

Beginning with the eighties *structured systems' analyzing and developing methods and methodologies* have been worked out and complex computer-aided tools have been developed, which are more then simple testing tools: they provide help for transforming the logical model of the system into physical. Further details about such methodologies and tools are described in section 5.1.

A next step in system-development is *the use of embedded intelligence.* Programs are "living" in the electronic circuits - they become organic elements of machines. Testing such programs obviously needs techniques that completely differ from ordinary testing techniques. Working out such testing techniques and tools is a new challenge for informaticians.

2.2. Five quality definitions. Let's imagine [Genuch91] that a product has been developed according to the specification, but the users are not satisfied because the product does not fit their needs. Developers conclude that the users are not able to explain what they want, the users conclude that developers are not able to understand what the user needs. Both parties end up being unsatisfied with the result of the development effort and it is not easy to say who is right and who is wrong. In the opinion of Van Genuchten "both parties are right, according to their own definitions".

Here are the 5 quality definitions given by [Garvin84]:

- The transcendent definition says that quality is absolute and universally recognizable, despite the fact that it cannot be defined precisely. Only experience can teach to recognize quality.
- According to the user-based definition, quality is "fitness for use". This definition starts from the assumption that the individual customers have different needs and those goods that best satisfy their needs are considered to have the highest quality.
- The product-based definition views quality as a precise and measurable variable.
- Manufacturing-based definition identifies quality as conformity with specifications.
- Value-based definition defines quality in relation to costs. A quality product provides performance at an acceptable price or conformance at an acceptable cost.

The product-based definition of quality is one of the most used. This definition deals with objective quality attributes and their relationship. Quality attributes - like correctness, reliability, efficiency, integrity, usability, maintainability, testability, flexibility, portability, reusability, interoperability - are defined, their interaction is shown by tables of correlance[2]. We can observe that using *the product-based quality definition involves some effective measurements on the software product* (e.g., number of comments in a program, number of errors, number of failures, etc.). This approach is now somehow neglected due to the new approach suggested by a popular standards series.

3. Today's fashion: ISO 9000. Among the many national and international quality organizations ISO[3] has undoubtedly reformed the concept of quality (the direction of this reform can be questionable). In March 1987 it issued the ISO 9000 Series, prepared by ISO/TC 176. In November 1987 CEN[4] adopted the set as EN 29000 Series - this was followed by a rapid adoption of the standards world-wide. The ISO 9000 News, March 1992 issue, lists 45 countries with identical adoption, and 3 countries - China, Jamaica, Venezuela - with equivalent adoptions, another booklet [Interl93] speaks about "over 90" countries that have adopted ISO 9000 by the beginning of 1993.

ISO 9000 series is a set of international standards for both *quality management and quality assurance*. ISO 8402 defines the concepts used, 9000 provides a guide for selection and use of the standards. Standards 9001 to 9003 deal with contractual situations, presenting three quality assurance models, while 9004 is in connection with non-contractual situations, presenting quality system elements used in quality management. The series has been completed with other standards, among which there is a standard for software quality management.

The ISO 9000 series was developed mainly for contractual business relationships. The goal is to increase customer confidence in the quality system used by their suppliers.

Theoretically there is no contradiction between "confidence in the quality system" and "confidence in the product's good quality", because a well designed and well maintained

[2] Quality attributes and their relationship have been modelled in the USA for the first time, by Boehm (1977) and by McCall (1978).

[3] International Standard Organisation

[4] Comité Européen de Normalisation

quality system should assure the proper quality of the products.

However, there is a contradiction between the two mentioned aspects, due to the fact that *the application of ISO 9000 suggests the use of manufacturing-based definition of quality, the development process being more emphasized, while measurable quality attributes suggest the use of the product-based definition.*

One of the most important features of ISO 9000 is that it provides a third-party auditing model to review, certify and maintain certification of organizations.

3.1. The main concepts of ISO 9000. ISO 9000 Series operates with many concepts, among which some important concepts are:

• Quality = the totality of features and characteristics of a product or service that bear on its ability to satisfy stated or implied needs[5].

• Quality policy = the overall quality intentions and direction of an organization as regards to quality, as formally expressed by top management.

• Quality management = that aspect of the overall management function that determines and implements the quality policy.

• Quality management system = the organizational structure, responsibility, activities, capabilities and resources that together aim to ensure that (software) products will satisfy stated or implied needs.

• Registration/ certification = the assessment of a company's quality system by a third party - a quality system registrar.

3.2. Registration. The company contacts such a registration body, which assesses first the documents of the quality system. In case the documents satisfy the requirements stated in the appropriate standard(s) of ISO series, an internal audit follows. The internal audit examines the compliance of the things written in the documentation with the things that really happen at the company. In most cases these reviews will expose inconsistencies that will be included in the final report. After the adequate corrective actions the auditors can fully

[5] The following definition was suggested by [Geiger93]: **Quality** = Realised totality of characteristics and their values of an entity in relation to requirement for quality. The definition proposed at the meeting of the ISO TC 176/SC 1 (1991) was rejected with the argument that the whole world is used to taking the existing definition.

approve, conditionally or provisionally approve or disapprove the company's registration. A company is registered for a three-year period. During that time, the organization must maintain and improve the quality system that was certified.

Registration certifies the quality management system developed according to ISO 9000, not the outstanding quality of the products.

3.3. Evolution of ISO 9000. ISO 9000 Series has evolved using some earlier quality-related standards (MIL-Q-9858A, MIL-I-45208A, ANSI/ASQC A-3, DEF/STAN 05-21,22,24,25&29), among which some national standards were issued in the UK, USA, Canada, France, Germany, the Netherlands.

It is interesting to observe that Japan has not contributed with a national standard to ISO. According to [Stephens93], quality management is not invented in Japan, although many terms of the modern disciplines of quality have been borrowed from successful and innovative Japanese applications. None of these terms, concepts and techniques *represent quality system assessment, certification, registration.* In fact ISO 9000 Series was only adopted in 1991 as the JIS Z 9900 series into the Japanese national standard system, mainly dictated by the international harmony, trade and co-operation.

This shows that *quality can be produced without having a registered quality management system* - and again takes us to the idea that *ISO 9000 registration is not always the guarantee for outstanding quality.*

In opinion of many quality specialists, ISO 9000 is just the first step to Total Quality Management. It deals too much with documents, forgetting the aspects apart from assessment. A quality management system developed only in order to registrate the company cannot be viable. (See Brian Plowman's opinion in 4.2.1.)

REFERENCES

[Beiczer78] Beiczer, Ö., Szentes, J.: Felhasználói programok forráslista alapján történô minôsítése, Softtech sorozat, D26, SzKI, 1978.

[Boehm78] Boehm, B.W. Characteristics of software quality, TRW Series of Software Technology, Vol. 1, North Holland, 1978

[Develin93] Develin & Partners Management Consultants, Freeing the Victims - Achieving Cultural Change Through Total Quality, 1993.

[Feigen93] Feigenbaum, A.: EOQ World Conference, Helsinki, in: MMT Társasági Tájékoztató, II. évf. 12. sz. 1993. dec.

[Garvin84] Garvin, D.A.: What does 'product quality' really mean ?, Sloan Management review, Fall 1984

[Geiger93] Geiger, W.: Talk Show with ISO 8402 ? Contemplative thoughts about the new quality vocabulary, EOQ Quality 1/1993, pg. 13-17

[Genuch91] Genuchten, M.van: Towards a Software Factory. Ph.D. thesis, Eindhoven University of Technology, 1991, ISBN: 90 900 4119 2

[Györkös94] Györkös, J.: A support for auditing the software process, in: Austrian-Hungarian Seminar on Software Engineering, Klagenfurt, 7-8 April 1994, pg. 7.1.-7.4.

[Havass93] Havass, M.: A szoftvervizsgálat múltja és jövõje, II. Minõségi Hét, 1993. november 8-10., A konferencia elõadásai, II. kötet, pg. 85-101

[Iterl93] The ISO 9000 Guide, Interleaf, 1993

[McCall78] McCall, J. A.: The utility of software metrics in large scale software systems development, IEEE Second software life cycle management workshop, August 1978

[Oracase] Barker, R: Case*Method, Tasks and Deliverables, Addison-Wesley Publishing Company, Revised Edition, 1990

[Pereira93] Pereira, Z. L.: EOQ World Conference, Helsinki, in: MMT Társasági Tájékoztató, II. évf. 12. sz. 1993. dec.

[PRINCE91] Project Run In Controlled Environment , LBMS Delegate Notes, 1991

[SSADM88] Downs E., Clare P., Coe I.: Structured Systems Analysis and Design Method - Application and Context Prentice Hall International Ltd, 1988)

[Stephens93] Stephens, K.: Quality Systems and Certification - Some Observations and Thoughts, EOQ Quality, 1/1993, pg 5-12

[Szentes82] SOMIKA - An automated System for Measuring Software Quality, Proc. of Premier Colloque de Génie Logiciel, AFCET, 261-276 (1982)

[Szentes83] Szentes, J.: Qualigraph - A Software Tool for Quality Metrics and Graphic Documentation, Proc. of ESA/ESTEC Software Engineering Seminar, Noordwijk, The Netherlands, 73-81, 1983.

[Szentes85] Szentes, J.: A szoftverminôség és mérése, Számítástechnika-alkalmazási Vállalat, Budapest, 1985

[TickIT90] Guide to Software Quality Management System Construction and Certification using ISO 9001/EN 29001 Part 1 (1987). Issue 1.1 (30. September 1990) British Computer Society UK Department of Trade and Industry

[ISO9000] Quality management and quality assurance standards - Guidelines for selection and use, 1987

[ISO 9001] Quality systems - Model for quality assurance in design/development, production, installation and servicing, 1987

[ISO 9004] Quality management and quality system elements - Guidelines, 1987

[ISO9000-3] Quality management and quality assurance standards - Guidelines for the application of ISO 9001 to the development, supply and maintenance of software, 1991

STUDIA UNIV. BABEŞ-BOLYAI, MATHEMATICA, XL, 3, 1995

PERTURBATION METHODS WITH LIE SERIES

Alin BLAGA[*]

Received: 3.09.1995
AMS subject classification: 68Q25, 68Q40, 65Y25

REZUMAT. - Metode de perturbaţie cu serii Lie. Puterea programării pe obiecte, facilităţile de paralelism ale calculatoarelor moderne cât şi cele ale produselor program de calcul simbolic sunt instrumentele perfecte pentru algoritmii folosiţi în teoria hamiltoniană a perturbaţiilor, pentru prezicerea traiectoriilor sateliţilor atrificiali. Este vorba, aici, despre metodele cu serii Lie, datorate lui Hori, Deprit şi Kamel. A se nota că formalismul Lie a fost introdus de Gröbner şi poate fi găsit în Nayfeh, sau Giacaglia.

1. Introduction. In celestian mechanics the methods used to predict the artificial satellites trajectories with a big accuracy were changed from time to time, become more eficient, more precise. Thus, we find important names in this field such Poincaré.

In Hamiltonian Theory of Perturbations the ice was broken by introducing Lie series and transforms, according to Hori (1966), Garrido (1968) and Deprit (1969). The Lie formalism can be found in Nayfeh (1973) and Giacaglia (1972). The basic idea is to find the solution of perturbation problems using power series. The problem is of the type:

$$\frac{dy_i}{dt} = x_i(t, y_j) + \varepsilon f_i(t, y_j).$$

This is a differential equations system with unknown functions $y_i(t)$ and with perturbed terms $\varepsilon f_i(t, y_j)$, where ε is a small parameter showing that the perturbing terms are small with respect to the principal terms of x_i. When ε is zero the system is reduced to the unperturbed system:

$$\frac{dy_i}{dt} = x_i(t, y_j).$$

In real problems the perturbing functions are more sophisticated, so it is too difficult

[*] *"Babeş-Bolyai" University, Faculty of Mathematics and Computer Science, 3400 Cluj-Napoca, Romania*

to solve them by classical Runge-Kutta methods, the error propagation is too high in such a problem. But we find particular problems solved in this way, using also refined numerical methods (see [16]). Steifel introduces the Fourier expansion as a good reason to approximate the solution.

Transformation of variables expanded in power of a small parameter plays an important role in the theory of perturbation. Hori (1966) and Deprit (1967) have proposed two methods to build canonical transformations, depending on a small parameter and based on the consideration of Lie series and transforms.

This paper tries to use the Lie transforms technology developed by Kamel (1969), combined with approximation methods of orthogonal Chebyshev polynomials. A big advantage of this technology is that it allows parallelism.

This method has been implemented in Maple V, but for high solution accuracy it was too hard to handle it, because of many symbolical differeniation and ordering, and impossibility of parallelization; and out of run-time and out of memory space appeared. It has been also implemented in MACSYMA and SPASM (see [2]), but there were the same problems of run-time and exhaustive memory needs. First, we have:

$$\begin{aligned} T_n(x, y) = H_n(x, y) \quad &+ \sum_{j=1}^{n-1} \left(C_{n-1}^{j-1} [H_{n-j}(x, y), S_j(x, y)] + \right.\\ &\left. + C_{n-1}^{j} K_{j, n-j}(x, y) \right), \text{ where} \\ S_n &= i(Tr), \\ K_n &= p(Tr), \\ K_{j,i} = [K_i, S_j] &- \sum_{m=1}^{j-1} C_{j-1}^{m-1} [K_{j-m, i}, S_i]. \end{aligned}$$

$[f,g]$ is Poisson bracket operator, defined as:

$$[f, g](P, Q) = \sum_{i=1}^{n} \left(\frac{\partial f}{\partial Q_i} \frac{\partial g}{\partial P_i} - \frac{\partial f}{\partial P_i} \frac{\partial g}{\partial Q_i} \right).$$

We also have here:

$$i(f) = \int_0^y f(p, q)\, dq_1$$

$$p(f) = \int [f - i(f)]\, dq_1, \text{ where the Hamiltonian:}$$

$$F(P, Q, \varepsilon) = \sum_{i=0}^{\infty} \frac{\varepsilon^i}{i!} K_i (P, Q).$$

To avoid Poisson brackets, Kamel developed an algorithm based on Lie transforms, where

$$K_{j,i} = K_{j,i-1} + \sum_{m=0}^{i-j} C_{i-j}^{m} L_{m+1} K_{j-m-1,\, i-1} \text{ and}$$
$$L_m K(x) = H_m(x) \frac{\partial K}{\partial x}(x).$$

This short table shows the run-time for the sequential algorithm (C++, PAC++) and for the parallel (Athapascan) algorithm. The default values are $n = 100$ for matrix dimension and m for matrices:

m - sequential/ processors - parallel	**PAC++** sequential double floating point (sec)	**PAC++** sequential Rational (sec)	**Athapascan** parallel double floating point (sec)
4	31.66	44.16	28.44
5	42.53	88.33	36.23
30	492.12	4057.3	402.32
100	2328.8	51021	2331.9

In this paper we do not try to increase the Hamiltonian's order, but increase the accuracy with a given precision ($\approx$ 30) according to speed. With such accuracy we can find a more powerful approximation of Lie transform, we have maximum run-time speed, by its parallelism and no more exhausted workspace. It is amazing what we can do using power orthogonal series, then aliate with symbolical calculus to make almost perfect the ideea that the trajectory of an artificial satellite is more precise, more perfect even after hundred revolutions.

Most of technologies of perturbation analysis can be introduced by a short study of a sample algebraic equation. So, let us consider the following quadratic:

$$x^2 + \varepsilon x - 1 = 0 \tag{1}$$

Here ε plays a perturbing role for the solution. This quadratic has exact solutions:

$$x_{1,2} = \frac{-\varepsilon \pm \sqrt{4 + \varepsilon^2}}{2}.$$

If we expand them using Taylor series, for a small ε, we find:

$$x_1 = 1 - \frac{1}{2}\varepsilon + \frac{1}{8}\varepsilon^2 - \frac{1}{128}\varepsilon^4 + \dots$$
$$x_2 = -1 - \frac{1}{2}\varepsilon - \frac{1}{8}\varepsilon^2 + \frac{1}{128}\varepsilon^4 + \dots$$

The most important thing is that we have a very good approximation, even for a few given terms. Let see for $\varepsilon = 0.1$ what is going on:

$$\begin{aligned} x_1 &\approx 1 \\ &\approx 0.95 \\ &\approx 0.95125 \\ &\approx 0.95124921 \\ x_1 &= 0.951249219 \dots \end{aligned}$$

Let us have the new quadratic:

$$\varepsilon x^2 + x - 1 = 0. \tag{2}$$

Here ε plays a perturbing role for the solution. For $\varepsilon = 0$ we have only one root, $x = 1$ and for $\varepsilon \neq 0$ we find two roots. This is the most easies example of that we call ***singular*** perturbation problem. The problems that are not singular are ***regular.*** In reality, most of the problems are singular and those are the most interesting.

Performing a Taylor series expansion for those two roots, for a small ε, we have:

$$x_1(\varepsilon) = 1 - \varepsilon + 2\varepsilon^2 - 5\varepsilon^3 + \dots \tag{3}$$
$$x_2(\varepsilon) = -\frac{1}{\varepsilon} - 1 + \varepsilon - 2\varepsilon^2 + 5\varepsilon^3 + \dots \tag{4}$$

But the power series for x_2 starts with an ε^0 instead of the usual ε^{-1}, so a very good ideea is to make a coordinate transformation, thus the singular equation becomes a regular one. That is the mechanism of Lie transforms, to regularise the singular problem and more, to reduce the finding of solution to a standard power series expansion form problem.

For example let us rescale:

$$x = \frac{y}{\varepsilon}. \tag{5}$$

We now have the regular equation:

$$y^2 + y - \varepsilon = 0.$$

2. General theory of the algorithm

DEFINITION. Let $G \subseteq C$ be an open set and $\rho_i : G^n \to C$, $i = \overline{1, n}$ holomorphic functions. If $z = (z_1, z_2, \ldots, z_n) \in G^n$, then

$$D = \rho_1(z) \frac{\partial}{\partial z_1} + \rho_2(z) \frac{\partial}{\partial z_2} + \ldots + \rho_n(z) \frac{\partial}{\partial z_n} \tag{6}$$

is the Lie differential operator.

Notation. Let be the open set $G \subseteq C$ and f, $\rho_i : G^n \to C$, $i = \overline{1, n}$, holomorphics. Then

$$Df = \rho_1(z) \frac{\partial f}{\partial z_1} + \rho_2(z) \frac{\partial f}{\partial z_2} + \ldots + \rho_n(z) \frac{\partial f}{\partial z_n}, \tag{7}$$

$$D^0 f = f, \tag{8}$$

$$D^{n+1} f = D(D^n f). \tag{9}$$

DEFINITION. A power series of type

$$\sum_{n=0}^{\infty} \frac{t^n}{n!} D^n f(z) = f(z) + \frac{t}{1!} Df(z) + \frac{t^2}{2!} D^2 f(z) + \ldots$$

where $f : G^n \to C$ is an holomorphic function, $G \subseteq C$ is an open set, $t > 0$, is called Lie power series. Formally

$$e^{tD} f(z) = \sum_{n=0}^{\infty} \frac{t^n}{n!} D^n f(z). \tag{10}$$

THEOREM. *$G \subseteq C$ is an open set. Any x-dependent, holomorphic and indefinitely differentiable vector $F(x,\varepsilon)$ of the form*

$$F(x, \varepsilon) = \sum_{n=0}^{\infty} \frac{\varepsilon^n}{n!} F_n(x),$$

where ε has a perturbing role, can be expressed in the form of another holomorphic and indefinitely differentiable vector of power series, using $x = y + \sum_{n=1}^{\infty} \frac{\varepsilon^n}{n!} F_n(x)$ transform, where f_n: $G \to C$

Proof. Let be the indefinitely differentiable vector $F(x,\varepsilon)$ developed in power series of the form

$$F(x, \varepsilon) = \sum_{n=0}^{\infty} \frac{\varepsilon^n}{n!} F_n(x), \text{ with} \tag{11}$$

$$F_n(x) = \left[\frac{\partial^n}{\partial \varepsilon^n} F(x, \varepsilon) \right]_{\varepsilon = 0}. \tag{12}$$

If $x = x(y,\varepsilon)$, then the vector has the power series form such as

$$F(x,\ \varepsilon) = \sum_{n=0}^{\infty} \frac{\varepsilon^n}{n!} F_{<n>}(x)\ , \text{ where} \tag{13}$$

$$F_{<n>}(x) = \left[\frac{\partial^n}{\partial\varepsilon^n} F(x,\ \varepsilon) \right]_{\varepsilon=0,\ x=y} . \tag{14}$$

The relation between $\frac{\partial F(x,\ \varepsilon)}{\partial\varepsilon}$ and $\frac{dF(x,\ \varepsilon)}{d\varepsilon}$ was established using obsolete differentiation formula:

$$\frac{dF}{d\varepsilon} = \frac{\partial F}{\partial\varepsilon} + \frac{\partial F}{\partial x}\frac{dx}{d\varepsilon}. \tag{15}$$

It is obvious that if x is y independent, F_n is expressed using partialy differentiation, else totaly differentiations required, such as in (14).

At the next step we need to make the transform

$$x = \sum_{n=0}^{\infty} \frac{\varepsilon^n}{n!} f_n(\ y),\ \text{ where} \tag{16}$$

$f_0(y) = y$, according to Lie transform by the known generating functions (12). Thus, for the $x \to y$ and known F_n functions we can express $F_{<n>}$. This powerful mechanism can reduce a singular system of differential equations to a regular one.

We may now differentiate equation (16), and we find

$$\frac{dx}{d\varepsilon} = \sum_{n=0}^{\infty} \frac{\varepsilon^n}{n!} f_{n+1}(\ y),$$

which has no solitary y. Let be $W(x,\ \varepsilon) = \sum_{n=0}^{\infty} \frac{\varepsilon^n}{n!} f_{n+1}(\ y)$ and $f \equiv W$ we have the simple formula

$$W(x,\ \varepsilon) = \sum_{n=0}^{\infty} \frac{\varepsilon^n}{n!} W_{n+1}(\ y)\ . \tag{17}$$

(17) is called a Lie Transform. This is just a power series, but it has the most important role to perform the final Kamel Transform.

Combining (15) with (17) we obtain

$$\frac{dF(x,\ \varepsilon)}{d\varepsilon} = \frac{\partial F(x,\ \varepsilon)}{\partial\varepsilon} + W(x,\ \varepsilon)\ \frac{\partial F(x,\ \varepsilon)}{\partial x}. \tag{18}$$

Let us make the notation $L_W F(x,\ \varepsilon) \equiv W(x,\ \varepsilon)\ \frac{\partial F(x,\ \varepsilon)}{\partial\varepsilon}$ which is the Lie derivative. Applied to (18) it will have a more compact form

$$\frac{dF(x,\varepsilon)}{d\varepsilon} = \frac{\partial F(x,\varepsilon)}{\partial\varepsilon} + L_W F(x,\varepsilon). \tag{19}$$

Let substitute F in (19) by (11) and W also in (19) but for this time in (17)

$$\frac{dF(x,\varepsilon)}{d\varepsilon}\sum_{n=0}^{\infty}\frac{\varepsilon^n}{n!}F_{n+1}(x) + \\ + \sum_{n=0}^{\infty}\frac{\varepsilon^n}{n!}\frac{\partial F_n}{\partial x}(x)\sum_{n=0}^{\infty}\frac{\varepsilon^n}{n!}W_{n+1}(x). \tag{20}$$

Sorting terms of (19) we have for ε^n

$$\frac{1}{n!}\,\frac{1}{0!}\,\frac{\partial F_0(x)}{\partial x}W_{n+1}(x),$$

$$\frac{1}{(n-1)!}\,\frac{1}{1!}\,\frac{\partial F_1(x)}{\partial x}W_n(x),$$

$$\vdots$$

$$\frac{1}{0!}\,\frac{1}{n!}\,\frac{\partial F_n(x)}{\partial x}W_1(x).$$

Thus, one obtains

$$\frac{dF(x,\varepsilon)}{d\varepsilon} = \sum_{n=0}^{\infty}F_{n+1}(x) + \sum_{n=0}^{\infty}\varepsilon^n\sum_{k=0}^{n}\frac{1}{k!\,(n-k)!}\,\frac{\partial F_{n-k}(x)}{\partial x}W_{k+1}(x), \text{ or}$$

$$\frac{dF(x,\varepsilon)}{d\varepsilon} = \sum_{n=0}^{\infty}\frac{\varepsilon^n}{n!}\left[F_{n+1}(x) + \sum_{k=0}^{n}C_n^k\,\frac{\partial F_{n-k}(x)}{\partial x}W_{k+1}(x)\right]. \tag{21}$$

Making the notation $F_{n,1}(x) = F_{n+1}(x) + \sum_{k=0}^{n} C_n^k\, L_{k+1}\, F_{n-k}(x)$, where

$$L_k F(x) = \frac{\partial F(x)}{\partial x}W_k(x),$$

will have for (21)

$$\frac{dF(x,\varepsilon)}{d\varepsilon} = \sum_{n=0}^{\infty}\frac{\varepsilon^n}{n!}F_{n,1}(x). \tag{22}$$

Performing formal notation of F_n by $F_{n,1}$ and by induction, yields to

$$\frac{d^mF(x)}{d\varepsilon^m} = \sum_{n=0}^{\infty}\frac{\varepsilon^n}{n!}F_{n,m}(x),\quad m\geq 1,\ \text{where} \tag{23}$$

$$F_{n,m}(x) = F_{n+1,m-1}(x) + \sum_{k=0}^{n}C_n^k\,L_{k+1}\,F_{n-k,m-1}(x),\quad m\geq 1,\quad n\geq 0. \tag{24}$$

Here, $F_{n,0} \equiv F_n$ and $F_{0,n} \equiv F_{<n>}$. QED, only if we show that $F_{<n>}$ is holomorphic. But we know that

$$F_{n,1}(x) = F_{n+1}(x) + \sum_{k=0}^{n} C_n^k L_{k+1} F_{n-k}(x),$$

so it is obvious, according to F_i, i ≥ 0, that $F_{n,1}$ is an holomorphic function. Thus, using (24), $F_{0,n}$, $n \geq 0$ are also holomorphic functions.

Remark. The recursive relation of (24) can be visualized in the forward triangle

$$\begin{tikzcd}
F_0 = F_{<0>} \arrow[d] & & & \\
F_1 \arrow[r] \arrow[d] & F_{0,1} = F_{<1>} \arrow[d] & & \\
F_2 \arrow[r] \arrow[d] & F_{1,1} \arrow[r] \arrow[d] & F_{0,2} = F_{<2>} \arrow[d] & \\
F_3 \arrow[r] \arrow[d] & F_{2,1} \arrow[r] \arrow[d] & F_{1,2} \arrow[r] \arrow[d] & F_{0,3} = F_{<3>} \arrow[d] \\
 & & &
\end{tikzcd}$$

Remark. The recursive relation of (24) is the same one found by Deprit, except to L_k operators, substituting Poisson brackets, which is more eficient.

We introduce some theoretical fundaments of Chebyshev polynomials, now.

DEFINITION. We call Chebyshev polynomial of degree n, the function T_n: $[-1,1] \to [-1,1]$,

$$T_n(x) = \cos n \arccos x. \tag{25}$$

LEMMA. *If T_n is the n degree Chebyshev polynomial, then*

$$T_{n+1}(x) = 2x\,T_n(x) - T_{n-1}(x), \quad n \geq 1. \tag{26}$$

LEMMA. *The n degree Chebyshev polynomial, $T_n(x)$, has n zeros on* $[-1,1]$

$$x_k = \cos\left(\frac{2k+1}{2n}\pi\right), \quad k = \overline{0,\, n-1} \tag{27}$$

and $n+1$ *extremal points*

$$x_k' = \cos\frac{k\pi}{n}, \quad k = \overline{0,\, n}. \tag{28}$$

THEOREM (of Orthogonality). 1°. *Continuous case. Let be the scalar product* $(f, g) = \int_{-1}^{1} \frac{f(x)\,g(x)}{\sqrt{1-x^2}}\,dx$. *Then*

$$(T_i, T_j) = \begin{cases} 0, & \text{if } i \neq j \\ \pi/2, & \text{if } i = j \neq 0 \\ \pi, & \text{if } i = j = 0. \end{cases}$$

2°. Discrete case. Let be the scalar product

$$(f, g) = \sum_{k=0}^{m} f(x_k)\, g(x_k),$$

where x_j, $j = \overline{0, n}$ are the zeros of Chebyshev polynomials of $m+1$ degree. Then

$$(T_i, T_j) = \begin{cases} 0, & \text{if } i \neq j \\ \dfrac{m+1}{2}, & \text{if } i = j \neq 0 \\ m+1, & \text{if } i = j = 0. \end{cases}$$

Remark. Fourier coefficients c_i^* for an orthogonal system $\{\varphi_i\}_{i=0}^{m}$ and continuous function f are picked-up from the interpolating problem

$$\sum_{j=0}^{m} c_j^* \varphi_j(x_i) = f(x_i), \quad i = \overline{0, m} \text{ where} \tag{29}$$

$$c_i^* = \frac{(f, \varphi_i)}{\|\varphi_i\|^2}, \quad i = \overline{0, m} \tag{30}$$

Remark. Fourier coefficients for Chebyshev polynomials are

$$c_0 = \frac{\sum_{k=0}^{m} f(x_k)}{m+1}, \tag{31}$$

$$c_i = 2\,\frac{\sum_{k=0}^{m} f(x_k)\, T_i(x_k)}{m+1}, \quad i = \overline{1, m} \tag{32}$$

LEMMA. *Let be c_i the Chebyshev coefficients of a continuous function f: $[-1,1] \to R$ and c_i' the Chebyshev coefficients of its derivative. Then*

$$c_{i-1}' = c_{i+1}' + 2(i-1)\, c_{i-1}, \quad i = \overline{m-1, 1}, \tag{33}$$

$$c_m' = c_{m-1}' = 0.$$

3. Sequential computing of Lie matrix. The computing scheme is obtained by equations presented in last section

$$F_{n,m}(x) = F_{n+1,m-1}(x) + \sum_{k=0}^{n} C_n^k L_{k+1} F_{n-k,m-1}(x), \quad m \geq 1, \quad n \geq 0, \tag{34}$$

$$L_k F(x) = \frac{\partial F(x)}{\partial x} W_k(x), \quad k \geq 1,$$

$$F_{n,0} \equiv F_n, \quad F_{0,n} \equiv F_{<n>}$$

We find useful some complexity studies. We start here with the background algorithm:

```
INPUT.        Matrix dimension N=n+1, where n is fixed;
              Generating functions Fi(x), i=0,n;
              Transform functions Wj(x), j=0,n;
ALGORITHM.
Step1.             F(i,0) := F(i), i=0,n;
              {filling the first colum - initialisation step}
Step2.             F(n,n-i) := F'(i), i=0,n-1;
              {initialise the last row with derivatives
              of the first column}
Step3.        for i=0,n do
                for j=1,i+1 do
                  Sum := 0;
                  for k=0,i-j-1 do
                    Sum := Sum + C(i-j-1,k) *
                               W(k+1) * F(n-j,n-i+j+k+1);
                           {C(n,k) are binomial coefficients}
                  endf;
                endf;
                {complete upper triangle:}
                F(i-j-1,j+1) := F(i-j,j) + Sum;
                {keeping the result before making modifications :}
                if n-j-1 = 0 then O(j+1) := F(i-j-1,j+1);
                endif;
                {complete lower triangle with derivatives :}
                F(n-j-1,n-i+j+1) := F'(i-j-1,j+1);
                {The main idea, for a best comprehensive algorithm
                is that on generate indices pairs of the form,
                keeping this order :
                         (0,1)
                         (1,1), (0,2)
                         (2,1), (1,2), (0,3)
                         (3,1), (2,2), (1,3), (0,4)
                  Corresponding indices are F(i,j) → F(i-j-1,j+1)
                                            F'(i,j) → F(n-j,n-i).
                  }
OUTPUT.       O(i),i=0,n;
```

We must note that the triangular matrix become a dense matrix by keeping also the derivatives. Thus, the algorithm is more complex, but is more faster, according to the complexity study at this stage. The computation scheme look like, at this moment:

$$\begin{tikzcd}
F_0 = F_{<0>} \arrow[r] \arrow[d] & F_{-1,1} \arrow[r] & F_{-2,2} \arrow[r] & F_{-3,3} \arrow[r] & {} \\
F_1 \arrow[r] \arrow[d] & F_{0,1} = F_{<1>} \arrow[r] \arrow[u] \arrow[d] & F_{-1,2} \arrow[r] \arrow[u] & F_{-2,3} \arrow[r] \arrow[u] & {} \\
F_2 \arrow[r] \arrow[d] & F_{1,1} \arrow[r] \arrow[d] & F_{0,2} = F_{<2>} \arrow[r] \arrow[u] \arrow[d] & F_{-1,3} \arrow[r] \arrow[u] & {} \\
F_3 \arrow[r] \arrow[d] & F_{2,1} \arrow[r] \arrow[d] & F_{1,2} \arrow[r] \arrow[d] & F_{0,3} = F_{<3>} \arrow[r] \arrow[u] \arrow[d] & {} \\
{} & {} & {} \quad . & {} &
\end{tikzcd}$$

This is just the formal algorithm which is the basic form of MapleV used algorithm, but for *C*++ or Athapascan languages it is not possible to make formal derivatives. These languages, first for a sequential algorithm second for a parallel one, can increase the speed of the algorithm and it is implemented by choosing Chebyshev approximation methods. This idea make a faster algorithm and a more accurate computations according to Plauger studies and implementation on C Standard Library. The algorithm is changing now:

```
INPUT.      n for matrix dimensions;
            m is the number of Chebyshev coefficients;
            Generating functions Fi(x), i=0,n;
            Transform functions Wj(x), j=0,n;

ALGORITHM.
Step1.      for i=0,m-1 do
               FRes_i[j,0] := c_i[F_j]; j=0,n;
            endfor;

Step2.      for i=0,m-1 do
               Fres_i[n,n-i] := diff(c_i[F_j]); j=0,n;
            endfor;

Step3.      for i=1,n do
              for j=1,i do
               for l=0,m-1 do
```

```
          compute c_1 of FRes[i,j] function;
        endfor;
      Setting FRes[i,j];
    {keeping the result :}
    if i=j then O(i) := FRes(i,i);
    endif;
    {complete lower triangle with derivatives :}
    FRes(j,i) := FRes(i,j);

OUTPUT.     O_1(i), 1=0,m-1; i=0,n;
```

One Lie matrix is filled with Chebyshev coefficients and thus, the function F_{ij} is represented by its $c_k(i\ ,\ j\ )\ ,\ \ k\ =\overline{0,\ m}$ Chebyshev coefficients on each matrix. We may see that it is an exhaustive memory consumption, but we win on run-time.

In the first step we must fill the first column functions of the forward triangle by computing all their coefficients. And the same thing for the first row of derivatives.

In Step1 we compute all the (m-1) coefficients for each $F_j\ ,\ \ j\ =\overline{0,\ n}$ continuous functions, using the common formula:

$$c_k = \frac{2}{m+1}\sum_{k=0}^{m} f\ (x_k)\ T_i(x_k),\ \ i\ =\overline{1,\ m},$$

where x_k are the Chebyshev roots of the T_i polynomial. Step2 was developed in purpose of computing all derivatives of continuous functions $F_j\ ,\ \ j\ =\overline{0,\ n},$ according to Chebyshev coefficients computed at the Step1 and to formula:

$$c'_{k-1} = c'_{k+1} + 2(k-1)\,c_{k-1},\ \ k = \overline{m-1,\ 1},$$

$$c'_m = c'_{m-1} = 0,$$

where c'_k are the corresponding Chebyshev coefficients to derivative of the function represented by $c_k,\ \ k\ =\overline{0,\ m-1}.$

The main loop needs to compute all $c(i,j)$ coefficients only by using $(j$-1)th column and j^{th} row of each matrix. For one $c_l(i,j)$ element we have (see also the (25) formula):

$$c_l(i,j) = CE(c(i,j-1)) + \sum_{k=0}^{i-j} C_{i-j}^{k}\,CE(c(j-1,\ i-k-1))\ \ \forall\ i=\overline{1,n},\ j=\overline{1,i},\quad (35)$$

where CE is a function that evaluate one function by its Chebyshev coefficients.

The internal Chebyshev computations are done by using of trigonometric high accuracy mathematical functions, that are more exact that the Standard C Library Math trigonometric functions. Also, for the binomial coefficients we made a vector to keep all values for all 1 to n degree. This method increase the algorithm run-time to very high speed and the vector reach only $\frac{(n+1)(n+2)}{2}$ size.

3.1 Complexity by algorithm. Some changes in the computation triangle are required, from this point of view.

If all values are recorded using triangle model shown in the 7th page of this paper, the algorithm complexity depends most of all in accessing and unaccessing direct matrix transform and some high precision evaluations. So, let consider that the other subalgorithms, wich are not shown in the background algorithm, have zero-complexity. Everyone can see that these subalgorithms are low-level functions that performs that performs some computations we do not need to know how, right now.

Dereferencing the classical forward triangle it leads to the upper triangle

$$\begin{tikzcd}
F_{0,0} \arrow[r] \arrow[d] & F_{0,1} \arrow[r] \arrow[d] & F_{0,2} \arrow[r] \arrow[d] & F_{0,3} \arrow[d] \\
F_{1,0} \arrow[r] \arrow[d] & F_{1,1} \arrow[r] \arrow[d] & F_{1,2} \arrow[d] & {} \\
F_{2,0} \arrow[r] \arrow[d] & F_{2,1} \arrow[d] & {} & \\
F_{3,0} \arrow[d] & {} & & \\
{} & & &
\end{tikzcd}$$

But let follow the algorithm and find its complexity. It is obvious that we may consider all matrix accesings and the first study is done only by this point of view.

At the Step 1 on access only the first column to store generating known functions F_n. If we denote by α_1 the algorithm to this level, will have

$$\text{cplx}\ \ (\alpha_1)\ = (n+1)\left[1 + \frac{n}{2}(n+1)\right]. \tag{36}$$

Next, on Step 2, must initialize the last row of lower triangle by first column derivatives. So, denoting by α_2 this algorithm, on leads to

$$\text{cplx}\ \ (\alpha_2)\ = \frac{n}{2}(n^2+7n+4)+1. \tag{37}$$

The recursive equation access intensively the matrix, in Step 3, so we must split its complexity in three parts. Anyway, the final result obtained here is

$$\text{cplx}\ \ (\alpha)\ = \frac{n^5}{8} + 2n^4 + \frac{43n^3}{8} + 12n^2 + \frac{21n}{2} + 7.$$

We don't need to worry about, because nothing will stop here. We must try another method to store and compute triangle computations. On see that we begin with storing not in the lower triangle, but in the upper side, wich is the natural ideea. But we access more frequent derivatives stored in the lower triangle. So, the complexity increase at very high order. Thus we must inverse storing. This leads to others formulae. First, the recursive formula is changing.

Thus, if we denote by β the new algorithm complexity, we have

$$\text{cplx}\ \ (\beta)\ = \frac{11n^4}{12} + \frac{5n^3}{2} + \frac{13n^2}{12} + \frac{n}{2} + 2.$$

This was obtained by the algorithm steps Step1, Step2 and Step3. It is obvious that cplx(β) << cplx(α). For example, if $n = 32$ on have

cplx(α) = 6,476,781 and

cplx(β) = 1,044,242.

Thus

$$\frac{\text{cplx}\ \ (\alpha)}{\text{cplx}\ \ (\beta)} = 6,202\,,$$

which means:

run-time α	**run-time β**
(sec)	(sec)
3600	580.27
1800	290.13
300	48.22
60	9.40
1.0	0.09

We can see its density and only one look make you think of its complexity. Even in this case, the method leads to a more efficient algorithm. Until this reached point we have made studies on the formal algorithm.

We discuss the β-version of the algorithm, because of its already studied performance. So, let be $\pi(t)$ the time need to access one vector element, $t(^{+})$, $t(^{*})$ the basic operations time needings for a user or predefined specified operands and $C(t)$ the time for Chebyshev computations.

Remark. If $C(i,j;t)$ is the complexity to compute $c(i,j)$ coefficients by using (35) then the main loop complexity is

$$\text{cplx}\ (t) = \frac{n(n+1)}{2} C(t) + (m+1) \sum_{i=1}^{n} \sum_{j=1}^{i} C(i\,,j\,;t)\,.$$

On evaluating cplx(t) let crossing $C(i,j;t)$ and find that

$$C(i,j;t) = (m+1)(i-j+2)\pi(t) + 3(i-j+1)\,t(^{*}) + t(^{+}) + (i-j+2)\,C(t) =$$

$$= (i-j+2)\,[\,(m+1)\,\pi(t) + C(t) + t(^{*})\,] + t(^{+}) - 3t(^{*})\,. \tag{38}$$

Thus the order of complexity will be of the form:

$$C(t) = \frac{n^3}{6} C(t) + \frac{n^3 m}{6} t(^{*}) + \frac{n^2 m}{2} t(^{+}) + \frac{n^3 m^2}{6} \pi(t)\,. \tag{39}$$

Remark. The complexity of one trigonometric function has the order

$$\tau(t) = 12\,[\,t(^{*}) + t(^{+}) + \pi(t)\,]\,.$$

4. Parallelization of the algorithm. An elementary task is defined as an indivisible work unit, specified in terms of its external environment such as I/O, execution time and so on. The parallel complexity study consists in splitting a part of the algorithm in elementary tasks. Thus, to compute F_{ij} function we need m Chebyshev coefficients. Thus the m tasks are:

$$\text{Task } P_t = \sum_{k=0}^{m} c^{(k)}_{i,j-1}\, T_l(c^{(t)}_{i,j}) - \frac{c^{(0)}_{i,j-1}}{2} + \sum_{k=0}^{m} c^{(k)}_{i,j}\left(\sum_{l=0}^{m} c^{(l)}_{j-1,\,l-k-1}\, T_l(c^{(t)}_{i,j}) - \frac{c^{(0)}_{j-1,\,l-k-1}}{2}\right) W(c^{(t)}_{i,j}),\quad t=\overline{0,m} \tag{40}$$

This leads to the well known precedence task graph, called *PARBEGIN - PAREND* graph, introduced by Dijkstra. If we denote by $t(T_i)$, the time to execute the task T_i, $i=\overline{0,m}$, then the T_1 time need to execute the algorithm from point $A(T_A)$ to point $B(T_A)$ will be:

$$T_1 = \max\{T_i,\ i=\overline{0,m}\},$$

instead of sequential time

$$T = \sum_{i=0}^{m} T_i .$$

We can write, using Dijkstra *PARBEGIN - PAREND* form, that

$$T_1:\ T_A;\ \mathit{PARBEGIN}\ T_0;\ |\,T_1;\ |\ldots\ |\,T_m;\ \mathit{PAREND};\ T_B.$$

In (35) *CE* can be use unless all $c^{(l)}_{i,j}$, $l=\overline{0,m}$ are computed. We must note that one computes for one function the all its coefficients into one FOR loop and these computations are independent. Also one coefficient needs a lot of time for the computations since this is dependent of mn^2 other coefficients. So this requires a small code for paralellization. One may see that we describe the parallel algorithm here.

```
INPUT. n for matrix dimensions;
       m is the number of Chebyshev coefficients;
       Generating functions Fi(x), i=0,n;
       Transform functions Wj(x), j=0,n;
```

```
ALGORITHM.
Step1.  for i=0,m-1 do
          FRes_i[j,0] := c_i[F_j]; j=0,n;
        endfor;
Step2.  for i=0,m-1 do
          FRes_i[n,n-i] := diff(c_i[F_j]); j=0,n;
        endfor;
Step3.  for i=1,n do
          for j=1,i do
            BEGIN PARALLEL LOOP from 0 to m-1
             compute c_l of FRes [i,j] function;
            END PARALLEL LOOP;
          Setting FRes[i,j];
        {keeping the result :}
        if i=j then O(i) := FRes(i,i);
        endif;
        {complete lower triangle with derivatives :}
        FRes(j,i) := Fres(i,j);
OUTPUT. O_l(i), l=0,m-1; i=0,n;
```

The *c(i,j)* element is computed by spawning all $c_l(i,j)$, from $l=0$ to $m-1$. This means, to this level, that on run with m parallel processes and at the end of all of them will have the *FRes(i,j)* function expressed in terms of Chebyshev polynomials.

The slave, in parallel algorithm, compute one coefficient for the specified function, so, if we want to obtain the maximum speed it will be preferable to set the numbers of slaves to the number of Chebyshev coefficients. The result is collected when all of the slaves done their work.

Parallelizing the algorithm leads to maximize relation (38) for $i = \overline{1, n}$ and $j = \overline{1, i}$. It is obvious that the order of $cplx_1$ is from now:

$$\mathcal{O}_1(t) = \frac{n^3}{2}\mathcal{C}(t) + \frac{n^3}{2}t(^{*}) + \frac{n^2}{2}t(^{+}) + \frac{n^2 m}{2}\pi(t). \tag{41}$$

So, getting the results from (39) and (40) on have

$$\mathcal{O}(t) - \mathcal{O}_1(t) = \frac{n^2 m}{2}\left(\frac{nm}{3} - 1\right)\pi(t) + \\ + \frac{n^2(m-1)}{2}t(^{+}) + \frac{n^3(m-1)}{6}t(^{*}) - \frac{n^3}{3}\mathcal{O}(t). \tag{42}$$

On 3D graphical representation by array dimensions and number of coefficients we can get the evaluating continuous time of all operations described here. We can see how time-smooth is the parallel computing behind the sequential computations. The smoothness can be also visualized according to (40).

REFERENCES

1. Blaga A., *Lie series and transforms for applications in celestial mechanics*, Institut IMAG, LMC Grenoble, France, 1995.
2. Char B., McNamara B., *Adiabatic invariants of simple Hamiltonian systems via the Lie transforms*, MACSYMA Users Conference, Washington DC, 1979.
3. Christaller M., *Athapascan-Oa sur PVM* 3, *définition et mode d'emploi*, IMA Grenoble, 1994.
4. Dahlquist G., Björck A., *Numerical Methods*, Prentice Hall, New Jersey, 1974.
5. Deprit A., *Canonical transformations depending on a small parameter*, Celestial Mechanics, volume 1, 1969.
6. Gautier T., Roch J.L., Villard G., *PAC++ v2.0, User and Developer Guide*, IMAG, Grenoble, 1994.
7. Gröbner W., *Die Lie-Reihen und ihre Anwerdungen*, Springer Verlag, Berlin, 1960.
8. Hamming R.W., *Numerical methods for scientists and engineers*, McGraw-Hill Book Company, New York, 1962.
9. Henrard J., *On a perturbation theory using Lie Transforms*, Celestial Mechanics, volume 2, 1970.
10. Kamel A.A., *Expansion formulae in canonical transformations depending on a small parameter*, Celestial Mechanics, Volume 1, 1969.
11. Kamel A.A., *Lie Transforms and the Hamiltonization of non Hamiltonian systems*, Celestian Mechanics, volume 4, 1971.
12. Kamel A.A., *Perturbation method in the theory of nonlinear oscillations*, Celestial Mechanics, volume 3, 1970.
13. Kinoshita H., *Third-order solution of an artificial-satellite theory*, Smithsonian Institution, Astrophisical Observatory, Cambridge, Massachusets, 1977.
14. Mersman W.A., *A new algorithm for the Lie Transformation*, Celestial Mechanics, volume 3, 1970.
15. Plauger P.J., *The Standard C Library*, Prentice Hall, New Jersey, 1992.
16. Stiefel P.J., Scheifele G., *Linear an Regular Celestial Mechanics*, Springer Verlag, New York, 1971.

STUDIA UNIV. BABEŞ-BOLYAI, MATHEMATICA, XL, 3, 1995

AN AXIOMATIC APPROACH TO ASSEMBLERS

Zoltán CSÖRNYEI*

Received: 30.07.1995
AMS subject classification: 68N05, 68N15, 68N20

REZUMAT. - O abordare axiomatică a asambloarelor. Scopul principal al asambloarelor este de a atribui valori simbolurilor din program. Programul obiect se asamblează din aceste valori. Se studiază procesul de atribuire a valorilor, valoarea simbolurilor se generează pe baza numărului de treceri şi a gradului de postdefinitate. Se determină numărul exact al trecerilor necesare traducerii programului de asamblare.

Abstract. - The main purpose of assemblers is to assign values to the symbols of an assembly program. The target program is assembled from these values. The process of assigning values is studied, the values of the symbols are generated depending on the number of passes and the degree of postdefinity. The exact number of passes needed to translate an assembly program is determined.

1. Introduction. The assembler creates the target program and the list from the assembly language program. The assembly language program is a series of symbols, during the translation the assembler assigns values to these symbols, and the target program and the list are composed from these values.

Emphasizing the common characteristics of assembly languages [1], [2], [7]-[11] firstly we define a grammar and the assembly language examined is generated by this grammar. The

* *Eötvös Loránd University, Department of General Computer Science, Budapest Hungary, E-mail: csz@maxi.inf.elte.hu*

symbols of the assembly language programs are classified depending on their positions. The syntactic error-free assembly language programs will be given by axioms, and The Fundamental Axiom of Assemblers [12] will be stated. In order to study the process of assigning values, the values of symbols are defined precisely and the classes of pre- and post-definit symbols are determined.

The degree of postdefinity [4] is introduced, and the relation between the value and the degree of postdefinity is studied to determine the minimal number of passes needs to translate the assembly program [3]. The result of this study is summarized in The Fundamental Theorem of Assemblers [5].

2. The assembly language. To study the assembly languages we define a simple context-free grammar G, and the assembly language program is a sentence of language $L(G)$ generated by grammar G. We give only the first few productions:

```
(1) <program>  → <line> eol |
                 <program><line> eol
(2) <line>     → <symb> = <expr> |
                 <stmt>
(3) <stmt>     → <symb> : <stmt> |
                 <mnem> |
                 <mnem><opnd>
(4) <opnd>     → <expr> |
                 <opnd> , <expr>
(5) <mnem>     →  ADD|MOV|...
....  ...          ...
```

From the above productions it is obvious that the unit of the assembly language program is the *line*, and that the line can be decomposed into three fields: label, instruction and operand field. The instruction field contains either a *directive* (e.g.=) or a *mnemonic* (e.g. ADD, MOV).

We complete the line with the serial number field. Thus, the k-th line of the assembly language program has the following structure:

(0)	(1)	(2)	(3)
k	*label*	*statement*	*operand*

With the production (3), it is possible to give labels in the program, and it can be seen that it is possible to have more labels for one mnemonic:

$symb_1$: $symb_2$: ... $symb_m$: *mnemonic operand* eol

The assembler assigns the actual value of the *current location counter* to the label. Marking this counter by \$, we transform the above line into the following form:

```
symb₁  =    $     eol
symb₂  =    $     eol
...
symbₘ  =    $     eol
 mnemonic operand  eol
```

After the transformation the assembly language program *P* consists of rows of the next types:

(0)	(1)	(2)	(3)
k		*mnemonic*	*operand*

k		*mnemonic*	

k	*symbol*	=	*operand*

Let $a(k)$, $m(k)$ and $u(k)$ denote the set of symbols in the field (1), field (2) and field (3) of the k-th line, respectively, and let $|P| = \max k$. Now we define the set of symbols:

DEFINITION 2.1 *Let* $A(0) = \{\$\}$, $A(k) = A(k\text{-}1) \cup a(k)$ $(1 \le k \le |P|)$, *and* $A = A(|P|)$. *Set A is called the parameter set of program P.*

We note that $\$ \notin a(k)$ $(1 \le k \le |P|)$.

Those $a \in A$ parameters which appear in the label fields in the line

(0)	(1)	(2)	(3)
k	a	=	$

are called *labels,* and let us denote by $b(k)$ the label of the k-th line:

$b(k) = \{a(k) \mid m(k) = \{=\} \wedge u(k) = \{\$\}\}$

The set of labels is the following:

DEFINITION 2.2 *Let* $B(0) = \emptyset$, $B(\mathrm{k}) = B(k\text{-}1) \cup b(k)$ $(1 \leq k \leq |P|)$, *and* $B = B(|P|)$. *Set B is called the* label set *of program P.*

In the assembly language program the labels need to be unique, that is a label cannot appear in other label fields of the program. This is stated by the following axiom.

Axiom I $b(k) \notin B(k\text{-}1)$, $1 \leq k \leq |P|$

If a label does not satisfy this axiom, then the assembler reports a "multiple defined symbol" error on every occurence of this label.

In the instruction fields there are directives or mnemonics:

DEFINITION 2.3 *Let* $M(0) = \emptyset$, $M(k) = M(k\text{-}1) \cup m(k)$ $(1 \leq k \leq |P|)$, *and* $M = M(|P|)$. *Set M is called the* instruction symbol set *of program P.*

Into the instruction field only an element of a previously given set $\mathcal{M}$ can be written. The set $\mathcal{M}$ is called the *instruction set of the assembly language.* The requirements for this set are stated by the following axioms:

Axiom II $M \subseteq \mathcal{M}$

Axiom III $A \cap M = \emptyset$

The symbols found in the operand fields are given in the following way:

DEFINITION 2.4 *Let* $U(0) = \emptyset$, $U(k) = U(k\text{-}1\} \cup u(k)$ $(1 \leq k \leq |P|)$, *and* $U = U(|P|)$. *Set U is called the* operand symbol set *of the program P.*

The $u(k)$ consists constants and operators, too, denote $u_C(k)$ and $u_O(k)$ the set of constants and the set of operators in the k-th line, and let U_C, U_O denote the set of constants and the set of operators of program P.

Similarly to the instructions, the elements of U_O can only be elements of a previously given set $\mathcal{O}$, and the restrictions for these sets are stated by the following axioms:

Axiom IV $U_O \subseteq \mathcal{O}$

Axiom V $U_O \cap A = \emptyset \wedge U_O \cap M = \emptyset$, *and*

$$U_C \cap A = \emptyset \wedge U_C \cap M = \emptyset$$

The non-operator and non-constant symbols found in the operand field must be defined in the label field, that is if

$$U_A = U \setminus (U_C \cup U_O),$$

then every element of U_A must be a parameter. This is stated in the fundamental axiom of assemblers.

Axiom VI (The Fundamental Axiom of Assemblers) $U_A \subseteq A$

Let $u_A(k) = u(k) \setminus (u_C(k) \cup u_O(k))$ $(1 \leq k \leq |P|)$. If for a symbol $s \in u_A(k)$ the above axiom does not hold, then the assembler in the k-th line gives an *"unknown symbol"* error message for the ymbol s.

3. The values of symbols. We will now examine what kind of values the assembler assigns to the elements of the previously defined sets of symbols. Let $val(s,k)$ denote the value of symbol s in line k. If the assembler cannot determine this value, then let it be $val(s,k) = \varepsilon$.

Axiom VII *For every symbol* $s \in U_C \cup M$, $1 \leq k \leq |P|$ *the inequality* $\mathrm{val}(s,k) \neq \varepsilon$, *and for every* $1 \leq k, l \leq |P|$ *the equality* $\mathrm{val}(s,k) = \mathrm{val}(s,l)$ *is valid.*

The above axiom states that the values of constants are determined unambiguously, and these values are independent from the lines. On the basis of the axiom the same is true for the mnemonic, that is the machine code assigned to the statement is independent from the program.

We only have to deal with the elements of sets A and U_O. As $\$ \in A$, we examine the value of this symbol first.

DEFINITION 3.1 *Let* val($,1) = 0, *and for* $k \geq 2$ *let*

$$\text{val}(\$,k) = \text{val}(\$,k\text{-}1) + f(m(k\text{-}1),u(k\text{-}1)),$$

where $f(m,u)$ is a non-negative integer function which gives the length of the machine code, its value for every $m(k$-$1) \subseteq M$ and for every $u(k$-$1) \subseteq U$ is previously given, even in the case of $s \in u(k$-$1)$, val$(s,k$-$1) = \varepsilon$.

According to the definition the value of the current location counter is zero in the first line, and the value of the increase is determined by the mnemonic and the symbols of the operand field, and not by the values of these symbols.

Let $A' = A \setminus \{\$\}$, $U_A' = U_A \setminus \{\$\}$, and let

$$u_A'(k) = \begin{cases} u_A(k) & \text{if } \$ \notin u_A(k) \\ u_A(k) \setminus \{\$\} & \text{if } \$ \in u_A(k) \end{cases}$$

In the following we deal with the symbols of A'. We can see that if $s \in A'$ is in the field (1), then in the operand field (3) only one expression can appear:

(0)	(1)	(2)	(3)
k	*s*	=	*expr*

The directive = means the assigning value, and for this reason we first give the value of the expresion.

We say that for an expression *expr* the inequality *val*(*expr*,*k*) $\neq \varepsilon$ is valid if

$u'_A(k) = \emptyset$ or if for all $s \in u'_A(k)$ the value of symbol s is known. Then

$$val\ (expr\ ,\ k) = \underset{u_0(k)}{\times}\ val\ (s,\ k),$$

where $s \in u_A(k) \cup u_C(k)$ and $\times$ means the execution of operations in $u_O(k)$.

If in the expression *expr* the $u'_A(k) = \emptyset$, then $val(expr,k) \neq \varepsilon$, as the expression at the very most can only contain symbol \$, constants and operators. The value of \$ according to Definition 3.1, the values of constants according to the Axiom VII are known.

A symbol s can occur in the label and operand field of the same line:

(0)	(1)	(2)	(3)
k	s	=	$s + 1$

and from this it can be seen that the value of the ymbol s in the two fields will not be the same. Let $val\ (s,\ k^{(j)})$ denote the value of symbol s in field j of the k-th line.

If $s \in a(k)$, then let $val\ (s,\ k^{(1)}) = val\ (expr\ ,\ k)$. We note if $u'_A(k) = \emptyset$, then $val\ (s,\ k^{(1)}) \neq \varepsilon$. If $val(expr,k) = \varepsilon$, then $val(s,k^{(1)}) = \varepsilon$.

As a symbol can appear in the label fields of different lines, we define the *scope* of symbols.

DEFINITION 3.2 *If* $s \in a(k_1),\ s \in a(k_2),\ \ldots,\ s \in a(k_m)$, *and* $1 \leq k_1 < k_2 < \ldots < k_m \leq |P|$, *then let the scope of symbol s* ($s \in a(k_i)$) *be*

$E(s,\ k_i) = \{(k_i + 1)^{(3)},\ (k_i + 2)^{(3)},\ \ldots,\ k_{i+1}^{(3)}\}$, *if* $i \neq m$, *and*

$E(s,\ k_m) = \{1^{(3)},\ 2^{(3)},\ \ldots,\ k_1^{(3)},\ (k_m + 1)^{(3)},\ \ldots,\ |P|^{(3)}\}$.

We note that for $s \in b(k)$ according to Axiom I

$E(s,\ k) = \{1^{(3)},\ 2^{(3)},\ \ldots,\ |P|^{(3)}\}$.

When we assign values to symbols, a very important property is the pre and postdefinity of symbols.

DEFINITION 3.3 *If* $s \in a(k)$ *and* $l^{(3)} \in E(s,k)$, *then symbol s in field* $l^{(3)}$ *in the case of*

$k < l$ *is said to be* predefinite, *in the case of* $k \geq l$ *is said to be* postdefinite *symbol.*

If $s \in a(k_1)$, $s \in a(k_2)$, ..., $s \in a(k_m)$, then symbol s in fields $i^{(3)}$, $2^{(3)}$, ..., $k_1^{(3)}$ is a postdefinite, and in fields $(k_1+1)^{(3)}$, ..., $|P|^{(3)}$ is a postdefinite, and in fields $(k_1+1)^{(3)}$, ..., $|P|^{(3)}$ is a predefinite symbol. Similarly, if label $s \in b(k)$, then label s in fields $1^{(3)}$, $2^{(3)}$, ..., $k^{(3)}$ is a postdefinite, and in fields $(k+1)^{(3)}$, ..., $|P|^{(3)}$ is a predefinite symbol.

From the last example, in which symbol s is in the label and the operand field, it is obvious that the value of the symbol s is depending on not only the fields but the passes, too. We extend the definition of values for passes, let $val^{(i)}(s, k^{(j)})$ denote the value of symbol s at pass i, in field j of the k-th line of the program.

For symbol \$ let

$$val^{(i)}(\$, k^{(3)}) = val(\$, k) \quad (1 \leq k \leq |P|)$$

where $val(\$,k)$ is the value given in Definition 3.1. At the begining of translation the value of symbol $s \in A'$ is unknown, that is for all $s \in A'$

$$val^{(1)}(s, 1^{(3)}) = \varepsilon,$$

and symbol s can only get value in such a first line, where in the label field appears:

DEFINITION 3.4 *If symbol* $s \in a(k)$ *and* $i \geq 1$, *then,*

$$\text{val}^{(i)}(s, k^{(i)}) = \begin{cases} \varepsilon & \textit{if } \exists\, r \in u_A(k), \\ & \textit{and } \text{val}^{(i)}(r, k^{(3)}) = \varepsilon \\ \underset{u_0(k)}{\times} \text{val}^{(i)}(q, k^{(3)}) & \textit{otherwise} \end{cases}$$

where $q \in u_A(k) \cup u_C(k)$, *and* $\times$ *means the execution of operation in* $u_O(k) \subseteq U_O$.

If we determine the value of a symbol, then the value in all of predefinite references is equal to this value, and the postdefinite references only get their values in the next pass.

DEFINITION 3.5 *If* $s \in a(k)$, $s \in u'_A(l)$, $l^{(3)} \in B(s, k)$ *and* $i \geq 1$, *then*

$$\text{val}^{(i)}(s, l^{(3)}) = \text{val}^{(i)}(s, k^{(1)}), \quad \textit{if} \;\; k < l, \;\; \textit{and}$$

$$\text{val}^{(i+1)}(s, l^{(3)}) = \text{val}^{(i)}(s, k^{(1)}), \quad \textit{if} \;\; k \geq l .$$

From these definitions unfortunately it can not be seen that the value of a symbol is determined in which pass. This is why we introduce the degree of postdefinity, and the relation between the value and the degree of postdefinity will be investigated.

4. The degree of postdefinity. The value of a parameter in the label field is undefined if in the operand field of the same line there is a symbol with undefined value. It is possible that the assembler gives a value to this symbol of the operand field in one of passes, and therefore in this pass and in the next passes the value of the parameter in the label field can be determined.

DEFINITION 4.1 *If $s \in a(k)$, then the* degree of postdefinity *of symbol s in $k^{(1)}$ let*

$$\text{pd}(s, k^{(1)}) = \begin{cases} 0 & \textit{if} \;\; u'_A(k) = \emptyset \\ \max\limits_{r \in u'_A(k)} \text{pd}(r, k^{(3)}) & \textit{otherwise} \end{cases}$$

According to the definition the symbol in the label field takes the highest degree of postdefinity from the operand field.

DEFINITION 4.2 *If $s \in a(k)$, $s \in u'_A(l)$ and $l^{(3)} \in E(s, k)$, then the degree* of postdefinity *of symbol s in $l^{(3)}$ let*

$$\text{pd}(s, l^{(3)}) = \begin{cases} \text{pd}(s, k^{(1)}) & \textit{if} \;\; l > k \\ \text{pd}(s, k^{(1)}) + 1 & \textit{if} \;\; l \leq k \end{cases}$$

The definition says that in postdefinite references the degree of postdefinity is one greater than in the line of its definition, and in predefinite refences the degree of postdefinity does not change.

If $s \in b(k)$, then $pd(s,k^{(1)}) = 0$, that is the degree of postdefinity in the label field of its definition line is zero, and according to the definitions, the degree of postdefinity in the predinite references equals zero and in the postdefinite references it equals one.

THEOREM 4.1 *If* $s \in a(k)$, *then* pd$(s,k^{(1)}) \geq 0$.

Proof. The statement is a consequence of Definition 4.1 and 4.2. □

COROLLARY 4.1 *From Theorem* 4.1 *and Definition* 4.2 *it follows that* pd$(s,1^{(3)})>0$ *for all symbols* $s \in u'_A(1)$.

THEOREM 4.2 *If* pd$(s, l_1^{(3)}) \neq$ pd$(s, l_2^{(3)})$, *where* $1 \leq l_1 < l_2 \leq |P|$, *then* $\exists k$, *for which* $l_1 \leq k < l_2$ *and* $s \in a(k)$.

Proof. If $l_1^{(3)}, l_2^{(3)} \in E(s, k)$, where $s \in a(k)$, then the theorem is a consequence of Definitions 4.1 and 4.2.

If $l_1^{(3)} \in E(s, k_1)$, and $l_2^{(3)} \in E(s, k_2)$, where $s \in a(k_1)$, $s \in a(k_2)$ and $k_1 \neq k_2$, then k_1, k_2 can not be elements of neither set $E_1 = \{1, ..., l_1-1\}$, nor set $E_2 = \{l_1, ..., l_2-1\}$, nor set $E_3 = \{l_2, ..., |P|\}$ at the same time, otherwise the inequality $pd(s, l_1^{(3)}) \neq pd(s, l_2^{(3)})$ is not hold. Similarly, the case of $k_1 \in E_1$ and $k_2 \in E_3$ is not hold. If $k_1 \in E_1$ and $k_2 \in E_2$, or $k_1 \in E_2$ and $k_2 \in E_3$, then the statement is true, since in the first case let $k = k_2$, and in the second case let $k = k_1$. □

Clearly not every symbol has a finite degree of postdefinity. For example, if symbol s only appears in the line

(0)	(1)	(2)	(3)
k	s	$=$	$s+1$

then according to Definitions 4.1 and 4.2

$$pd(s, k^{(1)}) = pd(s, k^{(3)}) = pd(s, k^{(1)}) + 1,$$

and we can state the following theorem:

THEOREM 4.3. *If* $s \in u'_A(k)$ *and* $k^{(3)} \in E(s, k)$, *then* pd$(s, k^{(1)}) = \infty$.

If $s \in u'_A(1)$, then Corollary 4.1 shows that $pd(s, 1^{(3)}) \geq 1$. If $pd(s, 1^{(3)}) < \infty$, then according to Theorem 4.3 $k \neq 1$ is true in the case of $1^{(3)} \in E(s,k)$, that is $\exists k$ $(1 < k \leq |P|)$, for which $1^{(3)} \in E(s,k)$. Thus, using Definition 4.2, we can state the following corollary:

COROLLARY 4.2 *If* $s \in u_A'(1)$ *and* $\mathrm{pd}(s, 1^{(3)}) = n < \infty$, *then in the case of* $1^{(3)} \in E(s,k)$ *it is true that* $k \neq 1$ *and* $\mathrm{pd}(s,k^{(1)}) = n\text{-}1$.

There is an obvious generalization of Theorem 4.3:

THEOREM 4.4 *If for symbols* $s_1, s_2, \ldots, s_m$

$s_1 \in a(k_1),\ s_2 \in u_A'(k_1),\ k_1^{(3)} \in E(s_2, k_2),$

$s_1 \in a(k_2),\ s_3 \in u_A'(k_2),\ k_2^{(3)} \in E(s_3, k_3),$

...

$s_m \in a(k_m),\ s_1 \in u_A'(k_m),\ k_m^{(3)} \in E(s_1, k_1),$

then $\mathrm{pd}(s_i, k_i^{(1)}) = \infty\ (1 \leq i \leq m)$.

Proof. The theorem follows from Definitions 4.1 and 4.2. □

We note that the infinite degree of postdefinity is based on a 'circle' in the declarations of symbols.

The next theorem states that if in the assembly language program there is a symbol with finite degree of postdefinity, then there is a symbol with zero degree of postdefinity too.

THEOREM 4.5 *If* $s \in a(k)$ *and* $\mathrm{pd}(s, k^{(1)}) < \infty$, *then* $\exists r \in A'$ *and* $\exists l$ $(1 \leq l \leq |P|)$, *for which* $r \in a(l)$ *and* $\mathrm{pd}(r,l^{(1)}) = 0$.

Proof. Let $pd(s, k^{(1)}) = i$, according to Theorem 4.1 $0 \leq i \leq \infty$. We prove the theorem by induction.

If $i = 0$, then let $r = s$, and $l = k$. Let us now suppose that the statement is true for every $j \leq i\text{-}1$, and we prove that the statement is true for $j = i$, too.

If $pd(s, k^{(1)}) = i$, then Definition 4.1 shows that $\exists r_1 \in u_A'(k)$, for which $pd(r_1, k^{(3)}) = i$, and if $r_1 \in a(k_1)$, then $k^{(3)} \in E(r_1,k_1)$. According to Theorem 4.3 $k_1 \neq k$. If k_1

$> k$, then according to Definition 4.2 $pd(r_1, k_1^{(1)}) = i-1$, and due to the hypothesis of the induction the statement is true.

If $k_1 < k$, then by Definition 4.1 it follows that $\exists r_2 \in u_A'(k_1)$, for which $pd(r_2, k_1^{(3)}) = i$, and if $r_2 \in a(k_2)$, then $k_1^{(3)} \in E(r_2, k_2)$. As $k_1 \neq k_2$, in the case of $k_2 > k_1$ the hypothesis of the induction is applicable, and if $k_2 < k_1$, then $\exists r_3 \in u_A'(k_2)$, for which $pd(r_3, k_2^{(3)}) = i$.

It can be seen that either we reach a symbol with degree of postdefinity i-1, or there exist

$$k > k_1 > \ldots > k_m$$

and $r_{m+1} \in u_A'(k_m)$, $pd(r_{m+1}, k_m^{(3)}) = i$. As $k_m \geq 1$, in the worst case $k_m = 1$. In this case however, due to Corollary 4.2, $pd(r_{m+1}, l^{(1)}) = i-1$, and therefore the hypothesis of the induction is applicable. □

From the above proof it can be also seen that if there is a symbol for which the value of the degree of postdefinity is equal to n $(0 < n < \infty)$, then there are symbols with degree of postdefinity $0,1,\ldots,n$-1:

COROLLARY 4.3 *If* $s \in a(k)$ *and* $\text{pd}(s,k^{(1)}) = n$, *where* $0 < n < \infty$, *then* $\exists r_1, r_2, \ldots, r_{n-1} \in A'$, *and* $\exists l_1, l_2, \ldots, l_{n-1}$, *for which* $1 \leq l_j \leq |P|$, $r_j \in a(l_j)$, *and* $\text{pd}(r_j, l_j^{(1)}) = j$ $(1 \leq j \leq n-1)$.

5. The relation between the value and the degree of postdefinity. We will now examine the relation between the value and the degree of postdefinity, and we will show that the degree of postdefinity of a symbol determines in which pass this symbol gets its value. Moreover, we prove that if this value was assigned in one of passes, then the value does not change in the following passes.

THEOREM 5.1 (The Fundamental Theorem of Assemblers)

If $s \in u_A^l(l)$ *and* $pd(s, l^{(3)}) = n < \infty$ *then* $\text{val}^{(n+1)}(s, l^{(3)}) \neq \varepsilon$.

Proof. We prove the theorem by induction. In the case of $n = 0$ the method of proof of Theorem 4.5 can be used, and it will not be presented here. Suppose that for every symbol $s_i \in u_A^l(k_i)$ in the case of $pd(s_i, k_i^{(3)}) = j-1$ the statement is true, that is $val^{(j)}(s_i, k_i^{(3)}) \neq \varepsilon$.

We prove that if $s \in u_A^l(l)$, $pd(s, l^{(3)}) = j$ is hold, then $val^{(j+1)}(s, l^{(3)}) \neq \varepsilon$.

If $s \in a(k)$ and $l^{(3)} \in E(s,k)$, then according to Theorem 4.3 $k \neq l$. If $k > l$, then using Definition 4.2 $pd(s, k^{(1)}) = j-1$, and on the ground of the hypothesis of the induction $val^{(j)}(s, k^{(1)}) \neq \varepsilon$. Definition 3.5 states that $val^{(j+1)}(s, l^{(3)}) = val^{(j)}(s, k^{(1)})$, thus the statement of the theorem is valid in the case of $k > l$

If $k < l$, then by Definition 3.5 it follows that the equality $val^{(j+1)}(s, l^{(3)}) = val^{(j+1)}(s, k^{(1)})$ is hold. As according to Definition 4.2 $pd(s, k^{(1)}) = j$, and Definition 4.1 shows that $\exists s_1 \in u_A^l(k)$, for which $pd(s_1, k^{(3)}) = j$. For this symbol s_1 $\exists k_1$, where $s_1 = a(k_1)$, $k^{(3)} \in E(s_1, k_1)$. If $k_1 > k$, then we have finished, according to Definitions 3.5 and 4.2 $val^{(j+1)}(s_1, k^{(3)}) = val^{(j)}(s_1, k_1^{(1)})$, and $pd(s_1, k_1^{(1)}) = j-1$.

If $k_1 < k$, then repeating the above procedure for s and k, either we reach a symbol with degree of postdefinity j-1, or we get series $s, s_1, s_2, \ldots, s_n$ and $k > k_1 > k_2 \ldots > k_n$. In the worst case $k_n = 1$, and composeing the next element s_{n+1} of the series, we find that $pd(s_{n+1}, 1^{(3)}) = j$. According to the Definitions 3.5, using Corollary 4.2, in the case of $s_{n+1} \in a(k_{n+1})$ $pd(s_{n+1}, k_{n+1}^{(1)}) = j-1$, and $val^{(j)}(s_{n+1}, k_{n+1}^{(1)}) = val^{(j+1)}(s_{n+1}, 1^{(3)})$. This value on the ground of the hypothesis of the induction $\neq \varepsilon$, which completes the proof. □

Now we prove that if we assign a value to a symbol in the $n+1$-st pass, then this value does not change in the following passes.

THEOREM 5.2 *Let* $s \in u_A^{l}(l)$. *If* $\mathrm{pd}(s, l^{(3)}) = n < \infty$, *then for all* $n_1, n_2 \geq n+1$ *the equality* $\mathrm{val}^{(n_1)}(s, l^{(3)}) = \mathrm{val}^{(n_2)}(s, l^{(3)})$ *is valid.*

Proof. We shall prove that $val^{(m)}(s, l^{(3)}) = val^{(m+1)}(s, l^{(3)})$ for every $m \geq n+1$, from this the statement of the theorem follows.

Suppose that the statement is not valid, that is the inequality $val^{(m)}(s, l^{(3)}) \neq val^{(m+1)}(s, l^{(3)})$ is true. We prove that from this condition it follows that there exists at least one symbol s_1, for which in the case of $s_1 \in a(k_1)$, $pd(s_1, k_1^{(1)}) \leq n-1$, and the values of symbol s_1 are different in two different passes.

Let $s \in a(k)$, $l^{(3)} \in E(s, k)$, then our condition is true according to Definition 3.5, if

1. in the case of $k > l$ $val^{(m-1)}(s, k^{(1)}) \neq val^{(m)}(s, k^{(1)})$, but using Definition 4.2, $pd(s, k^{(1)}) = n-1$, so we have found a symbol which has different values in different passes, and its value of degree of postdefinity equals to n-1. Let $s_1 = s$ and $k_1 = k$.
2. in the case of $k < l$ $val^{(m)}(s, k^{(1)}) \neq val^{(m+1)}(s, k^{(1)})$, and according to Definition 4.2, $pd(s, k^{(1)}) = n$. Using Definitions 3.4, 4.1 and 4.2, in the worst case more times repeating the above line of reasoning and by Corollary 4.2 it follows that we surely reach a symbol which has different values in different passes, an its value of degree of postdefinity equals to n-1.

Therefore $\exists s_1 \in u_A^{l}(k_1)$, for which $val^{(m-1)}(s_1, k_1^{(3)}) \neq val^{(m)}(s_1, k_1^{(3)})$ and $pd(s_1, k_1^{(3)}) \leq n-1$ in the case of m-1 $\geq$ n.

Using this method, we can produce a series of symbols $s_1, s_2, \ldots, s_{i_0}$ and a series of lines $k_1, k_2, \ldots, k_{i_0}$, for which $pd(s_{i_0}, k_{i_0}^{(1)}) = 0$, and $val^{(m)}(s_{i_0}, k_{i_0}^{(1)}) \neq val^{(m+1)}(s_{i_0}, k_{i_0}^{(1)})$. It is obvious that for the symbol s_{i_0}, with postdefinity degree zero, the above inequality is false, and thus we have proved the theorem. □

DEFINITION 5.1 *Let* pd(*P*) *denote the degree of postdefinity of program P*

$$\mathrm{pd}(P) = \begin{cases} 0 & \text{if } U_A' = \emptyset \\ \max\limits_{1 \le k \le |P|} \ \max\limits_{s \in u_A'(k)} \mathrm{pd}(s, k^{(3)}) & \text{otherwise} \end{cases}$$

The following theorem states that if the degree of postdefinity of a program is equal to n, then the program P can be translated by an n+1-pass assembler.

THEOREM 5.3 *For all symbols* $s \in U_A'$ *of program P in the case of* $s \in u_A'(l)$ $\mathrm{val}^{(\mathrm{pd}(P)+1)}(s, l^{(3)}) \neq \varepsilon$.

Proof. The statement is the consequence of Definitions 3.1, 3.4 and 3.5, as well as Theorems 5.1 and 5.2. □

As $U_A' \subseteq A$, according to the Definition 4.2 the statement of the above theorem is valid for every symbol $s \in A$.

COROLLARY 5.1 *For all symbols* $s \in A$ *of program P in the case of* $s \in a(k)$ $\mathrm{val}^{(\mathrm{pd}(P)+1)}(s, k^{(1)}) \neq \varepsilon$.

We have dealt with values of all symbols of an assembly program. We note that $pd(s, k^{(1)}) = 0$ in the case of $s \in b(k)$, and for $l^{(3)} \in E(s, k)$ $pd(s, l^{(3)}) \le 1$, the translation of labels can be done with a two-pass assembler, and this is the reason, that the most of assemblers has two passes. Thus, the most of assemblers unables to solve the translation of symbols of higher degree of postdefinity [6], and it is the programmers' responsibility to reduce the degree of postdefinity to 1, using modifications to the assembly language program, or simply with rearranging of lines.

REFERENCES

1. Brumm,P., Brumm,D., *8386/80486 Assembly Language Programming* (Windcrest/McGraw-Hill, 1993).
2. Chiu,P.P.K., Fu,S.T.K., A Generative Approacg to Universal Cross Assembler Design, *Sigplan Notices* Vol. 25(1990), No. 1. pp. 43-51.
3. Csörnyei Z.,: *Az assembláiás elmélete* (The Theory of Assemblers, in Hungarian) *Alkalmazott Matematikai Lapok* (to appear).
4. Csörnyei Z., On the Postdefinity of Symbols in Assembly Languages, *First Joint Conference on Modern Applied Mathematics*, Ilieni/Illyefava, Romania, 15-17 June, 1995. pp. 22-23.
5. Csörnyei Z., The Fundamental Theorem of Assemblers, *Conference at Dept. of Computer Science, Babeş-Bolyai University*, Cluj-Napoca, 8th November, 1994. (unpublished).
6. Csörnyei Z., "Univerzális mikroprocesszor assemblerek" (Universal microprocessor assemblers, in Hungarian), Doctoral Thesis Eötvös Loránd University, Budapest, 1980.
7. Giles,W.B., *Assembly Language Programming for the Intel 80xxx Family* (Macmillan Publishing Co., Inc., New York, NY., 1991).
8. Kindred,A.R., *Structured Assembler Language for IBM Microcomputers* (Hartcourt Brace, Orlando, FL., 1991).
9. Nelson,R.P., *Microsoft's 80386/80486 Programming Guide* (Microsoft Press. Redmond, WA., 1991).
10. Thorne,M., *Computer Organization and Assembly Language Programming for IBM PCs and Compatibles* (Benjamin-Cummings Publ. Co., Inc., Redwood City, CA., 1991).
11. Turley,J.L., *Advanced 80386 Programming Techniques* (Osborne McGraw-Hill, Berkeley, CA., 1988).
12. Wick,J.D., "Automatic generation of assemblers", Yale University, Dept. of Computer Science, Research Report No. 50., 1975.

STUDIA UNIV. BABEŞ-BOLYAI, MATHEMATICA, XL, 3, 1995

OPTIMIZATION OF d-CONVEX FUNCTIONS ON NETWORKS

Maria Eugenia IACOB*

Received: 22.11.1995
AMS subject classification: 05C12, 52, 05C35

REZUMAT. - Optimizarea funcţiilor d-convexe pe reţele. În acest articol sunt introduse şi studiate funcţiile d-convexe definite pe spatiul metric al unei reţele. Sunt discutate unele proprietăţi ale acestui tip de funcţii şi o metodă de rezolvare a problemei:

P: $f(z) \to \min,$

unde $f: N \to \mathbb{R}$ este o funcţie d-convexă.

1. Introduction. The actual period in the development of metric convexity is connected with investigations of discrete structures and of some extreme problems on them ([2], [14], [15], [13], [10], [22]). At the same time a considerable part of the results on convexity in discrete spaces is concentrated around metric convexity in graphs ([12], [16], [18], [20], [21]). It is interesting to mention that notions like convex set and convex function in graphs appeared previously in connection with some location problems ([3], [4], [5], [9], [23]). Another concept which was the direct result of location problems is the network (see [4], [5], [9]). In this article we deal with metric convexity (see [6], [7]) in networks and our aim is to define and study convex functions for these kind of spaces. We also give a method to solve the minimization of d-convex functions on networks. As we shall see, networks are closely related to graph, although they are not discrete metric spaces.

For convenience, we define here networks as metric spaces and some necessary notions related to them. Notice that we adopt definitions used in [3], [4], [5], [6], [7], [9].

* *"Babeş-Bolyai" University, Faculty of Mathematics and Computer Science, 3400 Cluj-Napoca, Romania*

We start with a undirected, connected graph $G=(W,A)$, without loops or multiple edges. To each vertex i in $W=\{1,...,n\}$ we associate a point v_i from X. Thus yields a finite subset $V=\{v_1,...,v_n\}$ of X, called the **vertex set** of the network. We also associate to each edge (i,j) in A a rectifiable arc $[v_i,v_j]\subset X$, called **edge** of the network. Let assume that $[v_i,v_j]$ has the positive length e_{ij} and denote by U the set of all edges. We define the **network** $N=(V,U)$ by the union $N = \bigcup_{(i,j)\in A} [v_i,v_j]$. It is obvious now that N is a geometric image of G, which follows naturally from an embedding of G in X. Let us suppose that for each $[v_i,v_j]$ in U there exists a continuous one-one mapping $Q_{ij}:[v_i,v_j]\longrightarrow[0,1]$ with $Q_{ij}(v_i)=0$, $Q_{ij}(v_j)=1$, $Q_{ij}([v_i,v_j])=[0,1]$ and if $x,y\in[v_i,v_j]$ such that $x\in[v_i,y)$ then $Q_{ij}(x)<Q_{ij}(y)$. It is obvious that to each point x from $[v_i,v_j]$ corresponds a unique point, namely $Q_{ij}(x)$, in $[0,1]$. Any connected and closed subset of an edge bounded by two points x and y of $[v_i,v_j]$ is called a **closed subedge** and is denoted by $[x,y]$. If one or both of x,y miss we say that the subedge is open in x (or in y) or is open and we denote this by $[x,y)$ or $(x,y]$ or (x,y), respectively. Using Q_{ij}, it is possible to compute the length of $[x,y]$ as $e([x,y])=|Q_{ij}(x)-Q_{ij}(y)|e_{ij}$. Particularly we have $e([v_i,v_j])=e_{ij}$, $e([v_i,x])=Q_{ij}(x)e_{ij}$ and $e([x,v_j])=(1-Q_{ij}(x))e_{ij}$.

By analogy with graphs we introduce the notions:

The **degree** $g_N(v)$ of $v\in V$ in N is the number of closed edges in N which contain v.

A **path** $D(x,y)$ linking two points x and y in N is a sequence of edges and at most two subedges at extremities. If $x=y$ then the path is called **cycle.** The **length of a path (cycle)** is the lengths sum of all its component edges and subedges and will be denoted by $e(D(x,y))$. If a path (cycle) contains only distinct vertices then we call it **elementary.**

A network N is **connected** if for any points x,y in N there exists a path $D(x,y)\subset N$.

An edge $[v_i,v_j]$ in U is called **isthmus** if $N\setminus(v_i,v_j)$ isn't connected.

Any connected subset $N'\subset N$ is called **subnetwork** of N. Any network $N'(V')=(V',U')$, where $V'\subset V$ and U' is the set of all edges from U having the extremities in V', is an **induced**

network.

A connected network without cycles is called **tree**.

Let $D^*(x,y)$ be the shortest path between the points x,y in N. We define a **distance** on N as follows: $d(x,y)=e(D^*(x,y))$ for any x,y in N. It is obvious that (N,d) is a metric space.

The **metric segment** between the points $x,y \in N$ is the set

$$<x,y>=\{z \in N \mid d(x,z)+d(z,y)=d(x,y)\}.$$

It is clear that the metric segment <x,y> coincides with the union of all the shortest paths between x and y.

A set $M \subset N$ is **d-convex** ([11]) if for any two points x,y in M we have $<x,y> \subset M$.

By **neighborhood** of the point $x \in N$ with radius r we mean the set

$$B(x,r)= \{z \in N \mid d(x,z)<r\}.$$

We also use as neighborhoods the sets $B_M(x,r)=\{z \in M \mid d(x,z)<r\}$, where $x \in M$ and M is some connected subset of N.

2. d-Convex Functions Our purpose in this section is to introduce the class of d-convex functions defined on the metric space (N,d) of a network N=(V,U). This approach was inspired by the papers [12], [17], [20], [21].

Let us consider a connected network N and a real valued function $f:N \longrightarrow R$.

Definition 2.1. ([16]) f is called d-convex on N if for any points $x,y \in N$ and any $z \in <x,y>$ the inequality

$$f(z) \leq \frac{d(x,z)}{d(x,y)}f(y)+\frac{d(y,z)}{d(x,y)}f(x)$$

holds.

One can state the following simple properties of d-convex functions on N. Note that this results was already proved for the more general case of metric spaces ([17], theorems 1-4).

Theorem 2.2. 1) For any d-convex functions f,g and any real number $\lambda \geq 0$, the functions f+g and λf are also d-convex.

2) The pointwise supremum of any family of d-convex functions is also a d-convex function.

3) The limit of any punctually convergent sequence of d-convex functions is also a d-convex function.

4) For any d-convex function f and any real number λ, the sets $\{z\in N \mid f(z)\leq\lambda\}$ and $\{z\in N \mid f(z)<\lambda\}$ are d-convex.

It will be needed the following preliminary results, which will establish links between d-convex functions and constants. Further on we denote by d-C and I the family of d-convex and respectively constant functions on N.

Lemma 2.3. If $C\subset N$ is an elementary cycle, then any d-convex function, $f:N\longrightarrow R$, is constant on C.

Proof. Let us consider the d-convex function $f:C\longrightarrow R$. What we have to prove is that for any $x,y\in C$, f(x)=f(y). It is easy to see that there exists the points $z_1,\dots,z_n\in C$, $n\geq 5$, satisfying the properties:

1. $z_i\in<z_{i-1},z_{i+1}>$, $i=2,\dots,n$, where $z_{n+1}=z_1$.
2. $z_1=x$ and there exists $k\in 2,\dots,n$, such that $z_k=y$.

Let us assume that $f(z_p)=\max\{f(z_i) \mid i=1,\dots,n\}$. From the d-convexity of f results

$$f(z_p) \leq \frac{d(z_{p+1},z_p)}{d(z_{p-1},z_{p+1})}f(z_{p-1}) + \frac{d(z_{p-1},z_p)}{d(z_{p-1},z_{p+1})}f(z_{p+1}) \leq$$

$$\leq \frac{d(z_{p+1},z_p)}{d(z_{p-1},z_{p+1})}f(z_p) + \frac{d(z_{p-1},z_p)}{d(z_{p-1},z_{p+1})}f(z_p) = f(z_p),$$

This leads to $f(z_p)=f(z_{p-1})=f(z_{p+1})$. By iterating this method we obtain $f(z_1)=\dots=f(z_n)$, thus f(x)=f(y). Since $x,y\in C$ were arbitrarily chosen, we conclude that f is constant on C. ■

The following definition refers to a class of networks, closely connected with d-convex functions.

Definition 2.4. A connected network $N=(V,U)$, is called quasitree if there exists at least one vertex $v\in V$ such that $g(v)=1$.

We mention below some simple properties regarding quasitrees that will be of use later.

Lemma 2.5. If $N=(V,U)$ is a quasitree with at least a cycle, then there is a connected subnetwork $R=(V',U')\subset N$, $V'\subset V$, $U'\subset U$, maximal with respect to inclusion, such that any vertex $v\in V'$ has $g_R(v)\geq 2$ and all cycles in N are contained in R.

Proof. Let us consider the set

$$V' = V'' \cup \Big(\bigcup_{v,v'\in V''} (D(v,v')\cap V) \Big),$$

where V'' is the set of all vertices in V, which lie on some cycle of N. The subnetwork generated by V', $R=N(V')$ is that one we are looking for. Indeed, from the way we define V', any $v\in V'$ has $g_R(v)\geq 2$. Consider now a cycle C in N. Then his vertices will be in V'' and hence $C\subset N(V')=R$. Now, let us prove that R is maximal. We assume that there exists a subnetwork $R_1=(V_1', U_1')\subset N$, having the same properties as R and $R\subset R'$. Consequently $|V_1\setminus V'|\geq 1$. We consider $v\in V_1\setminus V$ and $v_1,v_2\in V'$. From the way we define R it follows that there exists a path $D(v_1,v_2)\subset R$. On the other hand, since R' is connected, we deduce that there exists the paths $D(v,v_1)$ and $D(v,v_2)$, which are not contained in R.

It follows that the union $D(v_1,v_2)\cup D(v_2,v)\cup D(v,v_1)$ is a cycle of N not contained in R, which is a contradiction. ■

Remark. 1) Further on we refer to R as the root of the quasitree N.

2) If N does not contains any cycle, then any point from N can be viewed as R.

Lemma 2.6. The closure of $N\setminus R$, $cl(N\setminus R)$ is a not empty forest and each tree T from this forest satisfies $|T\cap R|=1$ and $T\cap R\subset V$.

Proof. This is the direct consequence of the previous lemma. Indeed $cl(N\setminus R)$ is a not necessarily connected network, without cycles, that is, a forest. From the definition of quasitrees results that $N\setminus R$ contains at least the vertex v of degree 1 and the edge incident to v, and therefore is not empty.

The fact that for any $T \subset N \setminus R$ holds $|T \cap R| = 1$ is also clear, since $|T \cap R| \geq 2$ implies the existence of a cycle not included in R. ■

Lemma 2.7. If $N=(V,U)$ is a quasitree and $f \in d\text{-}C$, then f is constant on the root R.

Proof. Considering Lemma 2.3 we can affirm that f is constant on any cycle in N. Consequently if two cycles C_1, C_2 have at least one common point then f is constant on $C_1 \cup C_2$. Taking into account the way we define the root of a quasitree and Lemma 2.5 it is clear that any two cycles in R either has notempty intersection or there exists a linking path between them. Our aim is to show that in this last case any $f \in d\text{-}C$ is constant on the union of this two cycles with the linking path. In order to get this consider two cycles C_1, C_2 and a path $D(x,y)$ such that $x \in C_1$, $y \in C_2$, $D(x,y) \cap C_1 = \{x\}$, $D(x,y) \cap C_2 = \{y\}$. If there exists another path $D(x',y')$ linking C_1 and C_2 then $C_1 \cup D(x,y) \cup D(x',y') \cup C_2$ will form a sequence of three or more cycles that can be ordered such that each two consecutive cycles have notempty intersection. This provides us that f is constant on $C_1 \cup D(x,y) \cup D(x',y') \cup C_2$.

Suppose now that $D(x,y)$ is the unique path between C_1 and C_2. Then $D(x,y) = \langle x,y \rangle$. Assume that $f(z) = \alpha_1$, for all $z \in C_1$, $f(z) = \alpha_2$, for all $z \in C_2$ and $\alpha_1 > \alpha_2$. Then by d-convexity of f for any $z \in D(x,y) \setminus \{x,y\}$ we have

$$f(z) \leq \frac{d(x,z)}{d(x,y)} f(y) + \frac{d(y,z)}{d(x,y)} f(x) = \alpha_2 \frac{d(x,z)}{d(x,y)} + \alpha_1 \frac{d(y,z)}{d(x,y)} < \alpha_1$$

On the other hand, for $r>0$, small enough, the set $cl(B(x,r))$ is a d-convex star. Let us consider $z_1 \in C_1 \cap cl(B(x,r)) \setminus \{x\})$ and $z_2 \in D(x,y) \cap cl(B(x,r)) \setminus \{x\})$. Obviously $x \in \langle z_1, z_2 \rangle$ and $f(z_2) < \alpha_1$. We have

$$\alpha_1 = f(x) \leq \frac{d(x,z_1)}{d(z_1,z_2)} f(z_2) + \frac{d(x,z_2)}{d(z_1,z_2)} f(z_1) =$$

$$= \frac{d(x,z_1)}{d(z_1,z_2)} f(z_2) + \alpha_1 \frac{d(x,z_2)}{d(z_1,z_2)} < \alpha_1 \frac{d(x,z_1)}{d(z_1,z_2)} + \alpha_1 \frac{d(x,z_2)}{d(z_1,z_2)} \alpha_1 = \alpha_1,$$

which is impossible. The same conclusion can be drawn for $\alpha_1 < \alpha_2$. Thus $\alpha_1 = \alpha_2 = f(z)$, for any z∈D(x,y). Thus f∈d-C is constant on R. ■

Summing up the above lemmas we conclude this part with

Theorem 2.8. d-C≠I if and only if N is a quasitree.

Proof. Consider a quasitree N=(V,U) and denote by R its root. Then any function f:N⟶R, f(x)=α+d(x,R), with $\alpha \in R_+$ is d-convex and obviously not constant. Let us prove that f is d-convex. We consider the points x,y∈N. The proof falls naturally in three parts.

1) x,y∈R. From Lemma 2.7 we have f(z)=α, for any z∈R and the inequality in Definition 2.1 holds.

2) x∈R and y∈N\R. Then for any z∈<x,y> we have

$$\alpha+d(z,R)=f(z)\le\frac{d(x,z)}{d(x,y)}f(y)+\frac{d(y,z)}{d(x,y)}f(x)=$$

$$=(\alpha+d(y,R))\frac{d(x,z)}{d(x,y)}+\alpha\frac{d(y,z)}{d(x,y)}=\alpha+d(y,R)\frac{d(x,z)}{d(x,y)}\Leftrightarrow$$

$$\Leftrightarrow d(z,R)\le\frac{d(x,z)}{d(x,y)}d(y,R)$$

If z∈R, then the previous inequality is true. If z∈N\R, then because z and y lies on the same tree T from cl(N\R) and T∩R={v} (see Lemma 2.6) we have

$$d(y,R)=d(y,z)+d(z,R),\ d(x,z)=d(z,R)+d(v,x).$$

Therefore

$$d(z,R)\le\frac{d(x,z)}{d(x,y)}d(y,z)+\frac{d(x,z)}{d(x,y)}d(z,R)\Leftrightarrow$$

$$\Leftrightarrow d(z,R)\left(1-\frac{d(x,z)}{d(x,y)}\right)=d(z,R)\frac{d(y,z)}{d(x,y)}\le\frac{d(y,z)}{d(x,y)}d(x,z)\Leftrightarrow$$

$$\Leftrightarrow d(z,R)\le d(x,z)=d(z,R)+d(z,x),$$

which is obvious.

3) x,y∈N\R. Then for any z∈<x,y> we have

$$f(z) \le \frac{d(x,z)}{d(x,y)} f(y) + \frac{d(y,z)}{d(x,y)} f(x) \Leftrightarrow$$

$$\Leftrightarrow \alpha + d(z,R) \le \frac{d(x,z)}{d(x,y)}(\alpha + d(y,R)) + \frac{d(y,z)}{d(x,y)}(\alpha + d(x,R)) \Leftrightarrow$$

$$\Leftrightarrow d(z,R) \le \frac{d(x,z)}{d(x,y)} d(y,R) + \frac{d(y,z)}{d(x,y)} d(x,R). \qquad (1)$$

At this point we have to analyze two cases:

i. $\langle x,y\rangle \cap R = \emptyset$. Then x,y lies on the same tree from cl(N\R). There exists $t \in \langle x,y\rangle$ such that for all $z \in \langle x,y\rangle$, $d(z,R)=d(z,t)+d(t,R)$.

Using this relation in (1) we have the sequence of equivalencies

$$(1) \Leftrightarrow d(z,t) + d(t,R) \le \frac{d(x,z)}{d(x,y)}(d(y,t) + d(t,R)) + $$
$$+ \frac{d(y,z)}{d(x,y)}(d(x,t) + d(t,R)) \Leftrightarrow$$
$$\Leftrightarrow d(z,t) \le \frac{d(x,z)}{d(x,y)} d(y,t) + \frac{d(y,z)}{d(x,y)} d(x,t) \Leftrightarrow$$

$$d(x,y)d(z,t) \le d(x,z)d(y,t) + d(y,z)d(x,t) \qquad (2)$$

Now we have to consider the possibilities:

a) $t=x$ $(t=y)$: (2) is equivalent with $d(x,y)d(z,t) \le d(x,y)d(z,t)$;

b) $z \in \langle x,t\rangle$: (2) is equivalent with $2d(x,z)d(y,z) \ge 0$;

c) $z \in \langle t,y\rangle$: (2) is equivalent with $2d(y,z)d(t,x) \ge 0$.

All these inequalities are true.

ii. $\langle x,y\rangle \cap R \ne \emptyset$. For any $z \in \langle x,y\rangle$ we have

$$f(z) \le \frac{d(x,z)}{d(x,y)} f(y) + \frac{d(y,z)}{d(x,y)} f(x) \Leftrightarrow$$

$$\Leftrightarrow d(z,R) \le \frac{d(x,z)}{d(x,y)} d(y,R) + \frac{d(y,z)}{d(x,y)} d(x,R). \qquad (3)$$

Since x and y lie on different trees from cl(N\R), the following relations hold:

If $z \in R$, then $d(z,R)=0$ and (3) is true.

If z lies on the same tree with x (respectively y) then $d(x,R)=d(x,z)+d(z,R)$ $(d(y,R)=d(y,z)+d(z,R))$ and therefore

$$(3) \Leftrightarrow d(z,R) \le \frac{d(x,z)}{d(x,y)}d(y,R)+\frac{d(y,z)}{d(x,y)}d(x,z)+\frac{d(y,z)}{d(x,y)}d(z,R) \Leftrightarrow$$

$$\Leftrightarrow d(x,z)d(z,R) \le d(x,z)(d(y,R)+d(y,z)) \Leftrightarrow$$

$$\Leftrightarrow d(x,z)(d(y,R)+d(y,z)-d(z,R)) \ge 0,$$

which is true, because $d(z,y) \ge d(z,R)$.

In order to prove the reverse implication we start by assuming that N is not a quasitree. Then all vertices in N are of degree at least 2. This allows us to affirm that any vertex is either on a cycle or on some path linking two cycles. But any d-convex function is constant on this kind of networks (see proof of Lemma 2.7) and hence d-C=I. ■

3. Optimization of d-convex functions

In this section we give a method to solve the problem of minimization without constraints, of a d-convex not constant function on a network. Many concrete problems are of this type. This becomes obvious if we refer to important location problems as the determination of centers and medians in networks (see [9]). On the other hand there are many problems where the constraints either do not influence the solution or are equalities and therefore can be reduced to problems or sequences of problems without constraints.

First we have to introduce two basic notions.

Definition 3.1. We said that a function $f:N \longrightarrow \mathbb{R}$ has a global minimum on N at the point $z \in N$ if for any point $y \in N$ we have $f(z) \le f(y)$.

Definition 3.2. We said that a function $f:N \longrightarrow \mathbb{R}$ has a local minimum at the point $z \in N$ if there exists a number $r>0$ such that $f(z) \le f(y)$, for any point $y \in B(z,r)$.

Let us recall (see [24]), that a metric space X is called Λ-convex, where $\Lambda \subset [0,1]$, if for every $x,y \in X$ and every $\lambda \in \Lambda$, there exists a point $z \in X$ such that $d(x,z)=\lambda d(x,y)$ and $d(z,y)=(1-\lambda)d(x,y)$. The following theorem is proved in [1] (also see [17], theorem 10).

Theorem 3.3. Let the space X be Λ-convex and $\lambda \in \Lambda$. If a d-convex function $f:X \longrightarrow \mathbb{R}$

has a local minimum on the d-convex set $A\subset X$, this minimum is also global.

It is easily seen that a network is a Λ-convex space and therefore Theorem 3.3 stands also for d-convex functions on networks.

On the other hand, because of Theorem 2.8 we have to consider only the case of quasitree, since this type of network is the only one which could be domain for a not constant d-convex function. Considering also the fact that any $f\in$d-C is constant on the root of a quasitree (see Lemma 2.7) we can state the following

Lemma 3.4. If N is a quasitree containing at least one cycle and $f:N\longrightarrow\mathbf{R}$ is d-convex, then any point from the root is a global minimum on N.

Proof. If N contains a contains a cycle then the root of N contains this cycle and therefore at least three edges. Taking in account Theorem 2.8, we can assume that $f(z)=\alpha$, for any $z\in R$.

Suppose now that there exists a point $x\in N\setminus R$ such that $f(x)<\alpha$. We denote by T that tree from cl(N\R), which contains x. We also consider an interior point z, of some edge included in R. If $\{v\}=\langle x,z\rangle\cap R\cap cl(N\setminus R)$ then by the d-convexity of f we have

$$\alpha=f(v)\le\frac{d(x,v)}{d(x,z)}f(z)+\frac{d(z,v)}{d(x,z)}f(x)=\alpha\frac{d(x,v)}{d(x,z)}+f(x)\frac{d(z,v)}{d(x,z)}<\alpha,$$

which is impossible. ∎

It is easy to see that any minimization of a d-convex function f, on a quasitree, can be reduced to the minimization of a function f' on the tree obtained from N by contracting the root R into a single point z_R. The function f' has the same values as f on the points from N\R and $f'(z_R)=\alpha=f(z)$, where $z\in R$. Then if S is the set of solutions for $f(z)\longrightarrow\min$ and S' is the set of solutions for $f'(z)\longrightarrow\min$ then clearly $S'\cup R\setminus\{z_R\}=S$. It is also important to observe that f' is also d-convex.

Thus we can conclude that it will be enough to find a method to solve the problem $f(z)\longrightarrow\min$, when N is a tree. In order to get such a method we need the following result

Theorem 3.5. If N is a tree, $f:N\longrightarrow\mathbf{R}$ is d-convex and S is the set of solutions for the problem

P: $f(z) \longrightarrow \min$

then S contains either a single point, or S is a subtree of N.

Proof. Let us assume that $|S|>1$ and consider two points $x,y \in S$. Then

$$\alpha = \min\{f(z) \mid z \in N\} = f(x) = f(y).$$

On the other hand from the d-convexity of f, for any $z \in \langle x,y \rangle$ we have

$$f(z) \leq \frac{d(x,z)}{d(x,y)} f(y) + \frac{d(y,z)}{d(x,y)} f(x) \leq \alpha.$$

It follows that $f(z)=\alpha$, for any $z \in \langle x,y \rangle$ and thus $\langle x,y \rangle \subset S$. Clearly, for any two points from S the metric segment between them is also contained in S. Thus S is a connected d-convex set of N, namely a subtree. ■

Remark. If we recall the previous proof it follows that the global minimum points set of is d-convex. Thus we recover a basic property of convex functions.

We are now able to give an algorithm to solve P.

Algorithm 3.6.

Step 1. Determine the set $VM = \min\{f(v) \mid v \in V\}$. Let $S = \emptyset$ and $\alpha = f(v)$, where $v \in VM$.

Step 2. Determine the set $UM = \{[v,v'] \in U \mid v \in VM\}$ and if $|UM| = k$, denote the elements from UM by $UM = \{u_1, \ldots, u_k\}$.

Step 3. For $j=1$ to k perform Step 4.

Step 4. Solve the problem:

P_j: $\min\{f(T_{u_j}(x)) \mid x \in [0,1]\}$,

where $T_{u_j} = Q_{u_j}^{-1}$. Let $\alpha_j = \min\{f(T_{u_j}(x)) \mid x \in [0,1]\}$ and S_j be the set of solutions for P_j.

If $\alpha > \alpha_j$ then $\alpha := \alpha_j$, $S := T_{u_j}(S_j)$ and go to Step 5.

If $\alpha = \alpha_j$ then $S := S \cup T_{u_j}(S_j)$.

Step 5. End algorithm with α as minimal value of f and S set of solutions for P.

Remark. 1) The problem P_j from Step 4, in the previous algorithm is a classic minimization problem of a convex function on [0,1]. Indeed for any $\lambda \in (0,1)$ and any $x,y \in [0,1]$ we have

$$f(T_{u_j}(\lambda x+(1-\lambda)y) = f(z_\lambda) \le \frac{d(z_x,z_\lambda)}{d(z_x,z_y)}f(z_y)+\frac{d(z_y,z_\lambda)}{d(z_x,z_y)}f(z_x) =$$

$$\frac{(Q_{u_j}(z_\lambda)-Q_{u_j}(z_x))e(u_j)}{(Q_{u_j}(y)-Q_{u_j}(x))e(u_j))}f(T(y))+\frac{(Q_{u_j}(z_y)-Q_{u_j}(z_\lambda))e(u_j)}{(Q_{u_j}(z_y)-Q_{u_j}(z_x))e(u_j)}f(T(x) =$$

$$\frac{\lambda x+(1-\lambda)y-x}{y-x}f(T_{u_j}(y))+\frac{y-\lambda x-(1-\lambda)y}{y-x}f(T_{u_j}(x) =$$

$$= (1-\lambda)f(T_{u_j}(y))+\lambda f(T_{u_j}(x)).$$

Taking also into account the analytic expression of $f \circ T_{u_j}$, we can use an appropriate technique of one dimensional minimization (see [8], p. 117-130).

2) The complexity of Algorithm 3.6. is $O(nO_1)$, where O_1 is the complexity of the method used to solve P_j.

3) There are situations when the difficulty of the problem will be increased by the determination of $f \circ T_{u_j}$, or this determination is technically impossible. In this case we propose the substitution of Step 4 with

Step 4'. Solve the problem:

P_j': $\min\{f(z)) \mid z \in u_j\}$.

Let $\alpha_j = \min\{f(z) \mid z \in u_j\}$ and S_j' be the set of solutions for P_j'.

If $\alpha > \alpha_j$ then $\alpha := \alpha_j$, $S := S_j'$ and go to Step 5.

If $\alpha = \alpha_j$ then $S := S \cup S_j'$.

For solving P_j' we propose the following approximation algorithm.

First we make the assumptions that $u_j = [v_j, v_j']$, $f(v_j) \le f(v_j')$ and $\epsilon = e(u_j)/p$, where p is fixed in order to obtain a satisfactory diminution of the error ϵ in finding the solution.

Algorithm 3.7.

Step 1. Set $x := v_j$, $y := v_j'$; xold:=x; yold:=y; $S_j' := \varnothing$; z is the middle point of u_j. If

```
f(x)=f(y)=f(z) then $S_j' := u_j$ and go to Step 6.
    Step 2. Repeat:
                If f(z)>f(x) then yold:=y and y:=z.
                Otherwise, if f(z)<f(x) then perform Step 3 and if f(z)=f(x)
                then perform Step 4.
        until ((d(x,xold)$\leq\epsilon$) and (d(y,yold)$\leq\epsilon$)) or (d(x,y)$\leq\epsilon$).
        Go to Step 5.
    Step 3. z':=z; z'':=z;
        Repeat:
                Assign the middle point of [x,z'] to z' and the middle point
                  of [y,z''] to z'',
        until (f(z')>f(z)) and (f(z'')>f(z)).
        xold:=x; x:=z'; yold:=y; y:=z''.
    Step 4. Assign the middle point of [x,z] to z'.
            If f(z')=f(z) then $S_j' := S_j' \cup [x,z]$; xold:=x; x:=z.
            Otherwise yold:=y; y:=z.
    Step 5. $S_j' := S_j' \cup [x,y]$;
    Step 6. Stop algorithm.
```

Remark. The previous algorithm is a combination of the bisection method and Fibonacci's technique.

REFERENCES

[1] T.T. Arkhipova, I.V. Sergienko, On the formalization and solution of some problems of organizing the computing process in data processing systems, Kibernetica 1973, 5, 11-18; Englsh transl. in Cybernetics 9 (1973).

[2] V. Boltyanskii, P. Soltan, Combinatorial geometry of various classes of convex sets, Ştiinţa, Chişinău, 1978. (Russian).

[3] P.M. Dearing, R.L. Francis,T.J. Lowe, A minimax location problem on a network, Transportation Sci., 1974, 8, 333-343.

[4] P. M. Dearing, R. L. Francis, T. J. Lowe, Convex location problems on tree networks, Oper. Res. 24

(1976), 628-624.
[5] J. Hooker, Nonlinear Network location Models, Ph.D. Thesis, Univ. of Microfilms Int., Ann Arbor, 1984.
[6] M.E. Iacob, V. Soltan, On geodetic Characteristic of Networks, Studii şi Cercetări Matematice, 5, 46, 1994, 521-527.
[7] M.E. Iacob, Divided differences and convex functions of higher order on networks, Studia Univ. Babeş-Bolyai Math., 3, 1994, 43-56.
[8] V. Karmanov, Programmation Matematique, Editions MIR, Moscou, 1977.
[9] M. Labbe, Essay in network location theory, Cahiers de Centre d'Etude et de Recherche Oper., 1985, 27(1-2), 7-130.
[10] T.T. Lebedeva, I.V. Sergienko, V.P. Soltan, On conditions for the coincidence of a local minimum and the global minimum in discrete optimization problems, Sov. Math. Dokl., 1985, 32(1), 78-81.
[11] K. Menger, Metrische Untersuchungen, Ergebnisse eines math. Kolloq. Wien. 1 (1931), 2-27.
[12] A. Sochircă, V. Soltan, d-Convex functions on graphs, Mat. Issled. No. 11(1988), 93-106. (Russian)
[13] P. Soltan, V. Cepoi, Solution of Weber's problem for discrete median metric spaces, Trans. Inst. Math. Tbilisi 85 (1987), 52-76. (Russian)
[14] P. Soltan, Ch. Prisăcaru, Steiner's problem on graphs, Dokl. Akad. Nauk SSSR 198 (1971), 46-49. (Russian)
[15] P. Soltan, D. Zambiţchi, Ch. Prisăcaru, Extremal problems on graphs and algorithms of their solution, Ştiinţa, Chişinău, 1973. (Russian)
[16] P. Soltan, V. Soltan, d-Convex functions, Dokl. Akad. Nauk SSSR 249 (1979), 555-558. (Russian)
[17] V. Soltan, Some properties of d-convex functions. I. II, Bull. Acad. Sci. Moldova. Ser. Phis.- Techn. Math. Sci. No. 2 (1980), 27--31; No. 1 (1981), 21-26. (Russian)
[18] V. Soltan, d-Convexity in graphs, Dokl. Akad. Nauk SSSR 272 (1983), 535-537. (Russian)
[19] V. Soltan, Introduction to the axiomatic convexity theory, Ştiinţa, Chişinău, 1984. (Russian)
[20] V. Soltan, Metric convexity in graphs, Studia Univ. Babeş-Bolyai. Mathematica (1991), No. 4, 3-43.
[21] V. Soltan, V. Cepoi, Some classes of d-convex functions in graphs, Dokl. Akad. Nauk SSSR 273 (1983), 1314-1317. (Russian)
[22] V. Soltan, V. Cepoi, d-Convexity and Steiner functions on a graph, Dokl. Akad. Nauk Belorussian SSR 29 (1985), 407-408. (Russian).
[23] B. C. Tansel, R. L. Francis, T. J. Lowe, Location on networks: a survey, Manag. Sci. 29 (1983), 482-511.
[24] J.S.W. Wong, Remarks on metric spaces, Nederl. Akad. Wetensch. Proc. Soc. A 69=Indag. Math. 28 (1966), 70-73.

STUDIA UNIV. BABEŞ-BOLYAI, MATHEMATICA, XL, 3, 1995

COLLECTIVE COMMUNICATION - A SOLUTION FOR RELIABLE NETWORK MANAGEMENT

Marius IURIAN**

Received: September, 4, 1995
AMS subject classification: 68M10, 68Q10.

Rezumat. - Comunicaţiile de grup - o soluţie pentru o gestiune a reţelelor cu grad ridicat de siguranţă. Lucrarea de faţă prezintă o soluţie pentru construirea unor aplicaţii de monitorizare şi gestiune a reţelelor de calculatoare care să fie tolerante la erori, cu un grad ridicat de siguranţă. Pentru aceasta se propune utilizarea comunicaţiilor de grup, într-o variantă derivată din modelul folosit de sistemul ISIS.

1. Introduction. The Simple Network Management Protocol (SNMP) helps network managers locate and correct problems in a TCP/IP network. Managers run a SNMP client on their local workstation and use the client to contact one or more SNMP servers that are running on remote machines. Implicit in the SNMP architectural model is a collection of network management stations and network elements. Network management stations execute management applications which monitor and control network elements. Network elements are devices such as hosts, gateways, terminal servers, and the like, which have management agents responsible for performing the network management functions requested by the network management stations. The Simple Network Management Protocol (SNMP) is used to communicate management information between the network management stations and the agents in the network elements.

All implementations of the SNMP must support the next five operations:

GetRequestPDU	Fetch a value from a specific variable
GetNextRequestPDU	Fetch a value without knowing its exact name
GetResponsePDU	Reply to a fetch operation
SetRequestPDU	Store a value in a specific variable

** *"Babeş-Bolyai" University, Faculty of Mathematics and Computer Science, 3400 Cluj-Napoca, Romania*

TrapPDU	Reply triggered by an event.

The first four operations are used to obtain or to set the value of the variables maintained by the SNMP servers and, in general, are less critical in time. For the Trap operation the time is very important because this operation is triggered by the occurrence of some specific event (for example a network card stopped functioning at time T). In fact the Protocol Data Unit (PDU) for the Trap operation contains a field that indicate the time of the event's occurrence.

The implementations of SNMP - servers or clients - are based on UDP (User Datagram Protocol) as a transport protocol. UDP provides connectionless communication among application programs. This make the protocol simpler and therefore with a greater performance as speed in comparison with a connection oriented protocol (like TCP - Transmission Control Protocol). The greatest disadvantage of UDP is that it is unreliable: it is possible to loose datagrams or these can arrive out of order or just after a great amount of time.

For these reasons we consider that the first four operations of SNMP can be implemented very well using UDP but an UDP based implementation of the Trap operation can be very inefficient and unreliable.

Other problems that should be mentioned here are the crash of the network management application or of the site which is running this application or the failure of the underlying network. These problems affect the availability of the network management service.

Another aspect that we want to mention before trying to evaluate different solutions is that the configurations of the sites that are being monitored and that are emitting SNMP Traps are sometimes difficult to modify. For example, a Novell Netware server running a SNMP server have only the possibility to send traps to a specified IP address, it cannot send this trap to a group of sites and cannot be modified very easy to do that (that modification implies the rewriting of the entire SNMP.NLM provided with the system, and this is not a trivial task).

2. **Different solutions.** The User Datagram Protocol (UDP) is unreliable if we consider an internetwork environment, but works very well in a local network environment (that is the principal reason for choosing UDP as transport protocol for the implementation of NFS - Network File System). This thing suggests the use of a **proxy** - a site which is in the same local area network as the monitored server. So, instead of instructing the server to send traps to a network management station, possibly situated outside the server's LAN, the server should send all the traps to this special site named proxy. The name was chosen because there is a great similarity with the proxies used by SNMP for the network objects that doesn't support directly SNMP (see [Stallings1993]).

Now that the traps sent by the servers arrived at the proxy, it is its responsibility to deliver the traps to the management station. A first approach can be to use a reliable transport mechanism (like TCP) to deliver the traps but this solution is still unable to overcome errors like site crashes and network failures. To achieve **availability** the network management station must be **replicated.** It is still possible to have the proxy responsible for sending copies of the trap to each of the network management station replica but it is important to remember that the ordering in time of such traps is very important. Also, the contents of the group of replicated management stations can vary in time (because of site crashes or network partitioning).

These are the reasons for choosing a model for **collective communications** in the implementations of the proxies and the management stations.

3. **The proposed Model.** The system is composed by a set of processes $P = \{ p_1, p_2, ..., p_n \}$ each one having a disjoint memory space. The processes represent the proxies and the network management stations. It is presumed that this set contains all the processes needed and it is known in advance. The processes failures follows the model **fail-stop**, which means that after the apparition of a failure the process stops all its activity (that means, more precisely, that in case of a failure a process stops immediately sending or receiving messages). The network can be partitioned due to link failures, messages can be lost, delayed, duplicated

or delivered out of order. The processes are structured in a set of process groups $G = \{ g_1, g_2, ..., g_m \}$. Each process group has a name, a set of component processes and a unique special process x which is named proxy (so each group is composed by a proxy, a number of network management stations and all their replicas) :

$\forall g_i \in G, g_i = \{ p_{i1}, p_{i2}, ..., p_{ik} \} \subseteq P.$

$\forall g_i \in G, \exists! p \in g_i$, p will be denoted proxy(g_i)

$X = \{ x \in P \mid \exists g \in G$ s.t $x=\text{proxy}(g)\} \subseteq P$ the set of proxies .

Any process can join or leave a group in any moment. A process take notice of a change in the contents of a group (to which it belongs) by using the notion of **view.** A view of a group is the list of its members. A **view-sequence** for a group g is an array: $\text{view}_0(g)$, $\text{view}_1(g)$, ..., $\text{view}_n(g)$ with the following properties:

$\text{view}_0(g) = 0$,

$\forall i$: $\text{view}_i(g) \subseteq P$,

$\text{view}_i(g)$ and $\text{view}_{i+1}(g)$ differs by exact a process.

A process take notice of a failure of some processes in the same group just by using this view-sequence. This model suppose that there always exist the possibility of direct communication between two processes. The transport level can offer two types of communications: multicast messages and point-to-point messages.

It is defined further the relation in time between different events (like sending and receiving messages) using the model proposed by L. Lamport in its very important paper "Time, clocks, and the ordering of events in a distributed system" (see [Lamport1978]).

DEFINITION 1: The process execution is a partially ordered sequence of events, each event corresponding to an atomic action. By " $\rightarrow_p$ " it will be denoted the acyclic order between two events that occur in process p.

The following notations will be used further in this paper:

$\text{send}_p(m)$	the event of sending the message m by the process p to one or more processes globally designed by dests(m).

$rcv_p(m)$	the event of receiving the message m by process p.
$rcv_p(view_i(g))$	the event by which process p take notice about the group g contents (group g including process p).
$deliver_p(m)$	the event of delivering the message m received before by process p.

If a process is a member of multiple groups then it must be indicated also the group to which a message is sent, received or delivered. The notation $deliver_p(m,g)$ means the delivering of message m to process p as a member of group g.

DEFINITION 2: The transitive closure of the relation "$\rightarrow p$" will be denoted "$\rightarrow$" and it will be an ordering relation having the following properties:

If $\exists$ $p \in P$ so that e "$\rightarrow p$" e' then $e \rightarrow e'$.

$\forall m$: $send_p(m) \rightarrow rcv_q(m)$.

Two distinct events a and b are **concurrent** if and only if we don't have neither $a \rightarrow b$ nor $b \rightarrow a$. For the messages m and m' the notation $m \rightarrow m'$ will have the significance of $send(m) \rightarrow send(m')$.

Many models represent the relation $\rightarrow$ using timestamp vectors.

DEFINITION 3: Let $VT(p_i)$ be the **timestamp vector** for a process p_i, an array of length n (where $n = |P|$) indexed by the process identifier. The rules for computing the timestamp vector are the following four:

$VT(p_i)$ is initialized with 0 when the process p_i starts.

For every event send(m) from p_k to p_i the component $VT(p_k)[i]$ is incremented by 1.

Every message sent by process p_i in multicast mode will contain the updated timestamp vector.

When a process p_k deliver a message m received from process p_i which contains the timestamp vector VT(m), will modify its own timestamp vector following the next rule:

$$\forall j \in \{1, 2, ..., n\}: VT(p_k)[j] := \max(VT(p_k)[j], VT(m)[j]).$$

So, the timestamp vector contained in a message counts the number of messages, calculated relative to each sending process, that causally precedes the message m. The timestamp vectors are compared using the following rules:

$VT_1 \le VT_2$ if and only if $\forall i$: $VT_1[i] \le VT_2[i]$

$VT_1 < VT_2$ if $VT_1 \le VT_2$ and $\exists i$: $VT_1[i] < VT_2[i]$

It can be easily proven that $m \rightarrow m'$ if and only if $VT(m) < VT(m')$.

The systems that implement group communication usually support three types of events ordering:

Causal ordering - which is the order defined before and represented using timestamp vectors.

Forced ordering - which means that a given sequence of events is occurring in the same order at every member of the group.

Immediate ordering - which means that every event is occurring in the same order at every member of the group, relative to every other event in the system.

It is obvious that the second order is stronger than the first and the third order is stronger than the second (and, of course, the second and the third orders are more difficult to implement, requiring more message exchanges between the processes).

ISIS system [Birman1991] supports all the three operations through three types of protocols: CBCAST, ABCAST and GBCAST and also the model proposed by Ladin-Liskov [Ladin1992] supports causal, forced and immediate operations. For our purpose it is sufficient the causal ordering and a protocol very similar to CBCAST protocol. In the fourth section we will indicate how this solution can be implemented using ISIS system version 2.1.

The protocol implements the following causal order:

(1) $\forall$ m, m', x $\in$ X: $send_x(m) \rightarrow send_x(m') \Rightarrow \forall$ p $\in$ dests(m) $\cap$ dests(m'):

deliver(m) $\rightarrow$p deliver(m').

The protocol for implementing causal ordering (given in formula (1)) in our case is described by the following rules:

Before sending the message m, the process $x_i \in X$ is incrementing $VT(x_i)[i]$ and insert the updated timestamp vector in message m.

The process $p_k \ne x_i$ ($p_k \notin X$) which receives the message m sent by the process x_i containing the timestamp vector VT(m), delays the deliver of m until the following condition become true: $\forall j \in \{1, 2, ..., n\}$ $VT(m)[j]$ =

$VT(p_k)[j] + 1$ if $j = i$ and $VT(m)[j] \leq VT(p_k)[j]$ otherwise.

When a message m is delivered the timestamp vector $VT(p_k)$ is updated according to the rules mentioned in definition 3.

Theorem: The protocol described before is correct - that means it respect the two properties of **safety** (it respect the causal order) and **liveness** (it will not infinitely postponing the deliver of any message).

Proof: Because the processes sending messages are members of X (the set of proxies) and the processes receiving messages are from $P \setminus X$ it result that messages sent by different proxies are not causally related. That means that for two distinct messages m_1 and m_2 sent by a process x_i (in this order) and arrived at process p_k will have the timestamp vectors in the following relation: $VT(m_1) < VT(m_2)$. Applying the second rule of the protocol we will have that message m_2 will be delivered only after the deliverance of the message m_1 and so the safety is proved.

For the proof of liveness let suppose that it exist a message m sent by the process x_i and that it cannot be delivered to process p_k. From the rule 2 of the protocol we will have one of the following relations true:

(2) $VT(m)[i] \neq VT(p_k)[i] + 1$.

(3) $\exists\, j \in \{1, 2, \ldots, n\}\ VT(m)[j] > VT(p_k)[j]$ where $j \neq i$.

The relation (2) means that m is not the next message to be delivered from x_i to p_k. Because the number of messages preceding m is finite results that exist a message m' sent by x_i, received by p_k and not yet delivered which the next message to be delivered (so m' satisfies the negation of (2)). If m' is also delayed we will obtain a contradiction.

Let consider now the relation (3) as being true. Let $n = VT(m)[j]$. The n-th transmission from process x_i must be a message m' which satisfy $m' \rightarrow m$ and which was either not received by p_k or it has been received and delayed. We can now repeat this reasoning for m' and because the number of messages transmitted before m' is finite and the relation $\rightarrow$ is acyclic it will result a contradiction.

4. Final remarks. The model proposed solve the raised problem in all its aspects offering a reliable solution for network management using SNMP Traps.

First of all it does not require any modifications on the monitored servers due to the use of proxies. The groups includes only the proxies and the network management stations

The replication of the network management stations and the use of a special protocol for collective communication make the solution reliable and performant.

For implementing the proxies and the network management applications it is possible to use ISIS the toolkit for distributed programming developed at Cornell University by a team lead by K. Birman [Birman1990]. The advantages of ISIS are the different types of groups organizations (peer-to-peer, client-server, diffusion and hierarchical) and the variety of protocols for group communication [Birman1993]. The presented model can be implemented using the diffusion type of group organization and the CBCAST protocol.

REFERENCES

[Birman1990]	K. BIRMAN, R. COOPER, T. JOSEPH, K. MARZULLO *"The ISIS System Manual"*, 1990.
[Birman1991]	K. BIRMAN, A. SCHIPER, P. STEPHENSON *"Lightweight Causal and Atomic Group Multicast"*, ACM TOCS, Vol. 9, No. 3, 1991, pp. 272-315.
[Birman1993]	K. BIRMAN *"The Process Group Approach to Reliable Distributed Computing"*, CACM, Vol. 36, No. 12, 1993, pp 36-54.
[Ladin1992]	R. LADIN, B. LISKOV *"Providing High Availability Using Lazy Replication"*, ACM TOCS, Vol. 10, No. 4, 1992, pp. 360-392.
[Lamport1978]	L. LAMPORT *"Time, Clocks, and the Ordering of Events in a Distributed Systems"*, CACM, Vol. 21, No. 7, 1978, pp. 558-565.
[Stallings1993]	W. STALLINGS *"SNMP, SNMPv2 and CMIP. The practical guide to Network-Management Standards"*, Prentice-Hall, 1993.

STUDIA UNIV. BABEŞ-BOLYAI, MATHEMATICA, XL, 3, 1995

UN MODELE DE REPRESENTATION D'OBJETS COMPLEXES AVEC IDENTITE D'OBJETS

Gr. MOLDOVAN[*] and C. BOBOILA[**]

Received: 9.10.1995

AMS subject classification: 68P15, 68R10

REZUMAT. - Un model de reprezentare a obiectelor complexe cu identitate de obiecte. Durant ces dernières années, les systèmes de bases de données orientés-objet ont émergé et sont en passe de devenir les principaux systèmes commerciaux des années 1995. Ces systèmes fournissent aux utilisateurs des possibilités de modélisation plus variées que les systèmes relationnels.

A partir de constructeurs tels que les constructeurs de n-uplet, ensemble, tableau, liste, et en imbriquant arbitrairement ceux-ci, de tels systèmes supportent la notion d'objet complexes avec identité d'objet.

Cet article présente un modèle de représentation d'objets complexes avec identité d'objet au moyen des graphes.

On définit les notions de graphe des types et graphe de composition d'objets pour mettre en évidence les connexions inter-objet.

Mots clés: Objets complexes, Valeurs structurées, Types structurés, Identité d'objet, Graphe des types, Graphe des types avec héritage, Graphe de composition d'objets.

1. Introduction

Des nouveaux domaines d'application émergent depuis quellques années. Ces domaines tels que: la conception assistée par ordinateur (CAO), la production de documents incorporant textes, images et graphiques, le génie logiciel, nécessitent la gestion d'une grande variété de types de données ainsi que les liens entre ces données (souvent imbriquées). Ces

[*] *"Babeş-Bolyai" Université, Faculté de Mathematique et Informatique, 3400 Cluj-Napoca, Romania*

[**] *Université de Craiova, Département de l'Informatique, 1100 Craiova, Romania*

données sont généralement appelées **Objets complexes.**

Le modèle relationnel [CODD.70] ne permet pas de modéliser efficacement ces types variés de données ainsi que leur imbrication. Dans ce modèle, les données sont représentées sous forme de relations "plates", c'est la contrainte de première forme normale. Les attributs d'un n-uplet étant nécessairement des valeurs atomiqurs (entiers, caractères, réels, booléens), il est difficile de représenter un objet complexe dans son ensemble.

D'autre part, le problème du dysfonctionnement entre langage de manipulation des données et lagages de programmation [ATKI.87] constitue un inconvénient supplémentaire de ces systèmes.

Plusieurs tentatives d'extension du modèle relationnel ont été effectuées au cours de ces dernières années [MAKI.77], [SCHE.82], [ZANI.85], [ABIT.86], [BANC.86].

Bien que ces modèles généralisent le modèle relationnel, ils ne permettent pas de fournir le partage d'objets par référence [KOSH.87]. Nous disons qu'un objet est partagé s'il est utilisé dans la construction d'un ou plusieurs objets. Cela signifie que l'espace des objets possède une structure de graphe orienté. Un modèle autorise le partage d'objet, s'il fournit la notion d'identité d'objet [KHOS.85].

Le rest du papier est organisé de la manière suivante: la Section 2 présente un bref aperçu sur les concepts utilisés pour le modèle O2, la Section 3 décrit le modèle à objets complexes avec identité d'objet, la Section 4 présente les définitions pour le graphe des types et le graphe de composition d'objets, nous concluons ensuite Section 5.

2. Le modèle O2: un bref aperçu

Nous présentons dans cette section le modèle d'objets complexes utilisé dans O2, qui est un système de bases de données orienté objet.

Bien que les solutions soient indépendantes de tel ou tel modèle (manipulant des objets complexes), nous utiliserons les notations du système O2 [LECL.88], [LECL.89], [ADIB.93], [BENZ.90], [BENZ.93].

Dans le système O2, deux types de concepts coexistent: les valeurs et les objets. Les valeurs possèdent un type, qui spécifie leur structure, et sont manipulées par des primitives prédéfinies. Les objets ont leur propre identité et encapsulent des valeurs ainsi que des méthodes définies par l'utilisateur. Les objets sont associés à classes.

Les classes sont identifiées par un nom unique. A chaque classe est associé un type ainsi qu'un ensemble d'objets.

Les types sont récursivement définis au moyen des constructeurs ensemble, liste et n-uplet et à partir de types de base tels que **integer**, **string**, etc. Certains types peuvent avoir leur existence propre et n'être pas associés à une classe.

L'exemple suivant illustre la construction de types ainsi que le lien entre classes et types.

```
add   class  Film
             type   tuple ( nom: string
                            année: integer
                            metteur-en-scene: Personne
                            acteurs: set (Personne))

add   class  Personne
             type   tuple ( nom: string
                            pays: string
                            profession: string)
```

L'ensemble de tous les objets (instances) d'une classe est appelé **extension** de la clase. Gérer une extension, pour une classe c, revient à créer, automatiquement, une hiérarchie d'héritage (ou relation de sous-typage) et les données et traitements (souvent appellés **méthodes**) sont encapsulés. Dans le modèle O2, la notion d'heritage multiple est définie [LECL.88], [LECL.89], [BENZ.93], [ADIB.93].

3. Modèle à objets complexes avec identité d'objet

Dans le système **O2**, chaque objet est identifié par un unique identificateur et représenté par un couple (i, v) où i est un identificateur et v une valeur. Nous rappelons, içi, qu'un objet peut être composé d'autres objets ou des valeurs. Un type est associé à chaque valeur.

Nous avons considéré les ensembles suivants:

- Un ensemble fini de domaines $D_1, \ldots, D_n$, n $\geq$ 1 (par exemple l'ensemble **Z** de nombres entiers). Notons **D** l'union des domaines $D_1, \ldots, D_n$. Nous supposons que les domaines sont disjoints.

- Un ensemble dénombrable et infini **A** , qui s'appelle **univers d'attributs**. Les éléments de **A** sont des noms pour les champs de la structure.

- Un ensemble dénombrable et infini **I d'identificateurs**. Les éléments de **I** seront utilisés comme d'identificateurs pour d'objets.

Les valeurs sont construites, récursivement, de la manière suivante:

Définition 1: **Valeurs**

Soit **A** un univers d'attributs et **D** un domaine de valeurs atomiques. Une valeur simple est prise dans l'un de types prédéfinis que sont les entiers (**integer**), les valeurs réelles (**real**), les valeurs logiques (**boolean**), les caractères (**char**) et les chaînes de caractères (**string**).

(i) Tout élément de **D** est une valeur (dite **atomique**).

(ii) Un identificateur d'objet est une valeur.

(iii) Si v_1 , ... , v_n sont des valeurs et a_1, ... , a_n des attributs de A alors

v = **tuple**(a_1: v_1,..., a_n : v_n) est une valeur structurée de type n-uplet.

(iv) Si v_1 , ... , v_n sont des valeurs distinctes alors

v = **set** (v_1 , ... , v_n) est une valeur structurée de type ensemble.

(v) Si v_1,...v_n sont des valeurs alors v= **list**(v_1,..., v_n) est une valeur structurée de type liste.

Notons **V** l'ensemble des toutes les valeurs.

Les objets sont récursivement construits de la manière suivante:

Définition 2: **Objets**

(i) Un objet est un couple o = (i , v) où i est un élément de I (un identifiant) et v est une valeur.

(ii) Si i , i_1 , ... , i_n sont des identificateurs d'objets et a_1 , ... , a_n sont des noms d'attributs à condition que $a_j \neq a_k$ pour tous les j, k de 1 à n , alors

o = (i , **tuple** (a_1 : i_1 , ... , a_n : i_n)) est un objet à valeur n-uplet.

(iii) Si i , i_1 ,... , i_n sont des identificateurs d'objets, alors

o = (i , **set** (i_1 , ... , i_n)) est un objet à valeur ensemble.

(iv) Si i , i_1 , ... , i_n sont des identificateurs d'objets alors

o = (i , **list** (i_1 , ... , i_n)) est un objet à valeur liste.

(v) Si nous notons **O** l'ensemble d'objets, alors O = I X V.

Il existe un atome particulier qui est noté **nil** et qui dénote un objet indéfini. Par abus de langage, si la valeur d'un objet est un atome, nous disons qu'il s'agit d'un **objet atomique**.

Une valeur composite se construit en utilisant les constructeurs n-uplet (**tuple**) , ensemble (**set**) et liste (**list**), appliqués récursivement. Si un objet a une valeur composite nous parlons alors d'**objet composite** ou **complexe**.

La figure 1 présente le diagrame pour la construction de valeurs composites pour des objets.

$$\begin{tikzcd}
 & \textbf{tuple} \arrow[loop above] \arrow[dl, leftrightarrow] \arrow[dr, leftrightarrow] \arrow[dd, leftrightarrow] & \\
\textbf{list} \arrow[loop left] \arrow[rr, leftrightarrow] \arrow[dr] & & \textbf{set} \arrow[loop right] \arrow[dl, leftrightarrow] \\
 & \textbf{atome} &
\end{tikzcd}$$

Figure 1. Construction de valeurs composites.

Les types sont récursivement construits de la manière suivante:

Définition 3: Types

Soit **A** un univers d'attributs et **T** un ensemble de types atomiques (**integer**, **real**, **boolean**, **string**, **char**, etc).

(i) Un atome est un type (dit **atomique**).

(ii) Les noms de classes sont des types.

(iii) Si t_1 , ... , t_n sont des types et a_1 , ... , a_n des attributs de A, alors

t = **tuple** (a_1: t_1, ..., a_n : t_n) est un type de structure n-uplet.

(iv) Si t_1 est un type alors t=**set**(t_1) est un type ensemble.

(v) Si t_1 est un type alors t=**list**(t_1) est un type liste.

(vi) Notons T l'ensemble des types.

De manière simple, un type t_1 est considéré comme un sous-type d'un type t_2, si toute instance de t_1 peut aussi être une instance de t_2.

Considérons tout d'abord un cas particulier très simple de sous-typage qui utilise les entiers. Notons n ... m le sous-type du type **integer** correspondant à l'ensemble des entiers de n à m bornes comprises.

On définit récursivement la relation de sous-typage p sur des sous-types de la façon suivante:

Définition 4: Sous-types

(i) $n .. m < p .. q$ si et seulement si $p \leq n$ et $q \geq m$.

(ii) Si $t_1,...,t_m,u_1,...,u_n$ sont des types et a_1, ... , a_m sont des attributs alors

tuple (a_1 : t_1 , ..., a_m : t_m) < **tuple** (a_1 : u_1 , ... , a_n : u_n) si et seulement si

$t_i < u_i$ pour tout i entre 1 et n et $n \leq m$.

(iii) Si t_1 , t_2 sont des types alors **set** (t_1) < **set** (t_2) si et seulement si $t_1 < t_2$.

(iv) Si t_1 , t_2 sont des types alors **list** (t_1) < **list** (t_2) si et seulement si $t_1 < t_2$.

Exemple

Soit le type t_1 avec la définition suivante:

```
t1 = tuple (
        nom: string,
        adresse: tuple (
                    numéro: integer,
                    rue: string,
                    ville: string,
                    code-postal: integer)
        téléphone: integer)
```

t_1 est un sous-type de t_2 qui est:

```
t2 = tuple (
        nom: string,
        adresse: tuple (
                    numéro: integer;
                    rue: string,
                    ville: string))
```

set (**tuple** (nom: **string**, age: **integer**)) est un sous-type de **set** (**tuple** (nom: **string**)).

list (**tuple** (nom: **string**, age: **integer**)) est un sous-type de **list** (**tuple** (nom: **string**)).

4. Graphe des types et graphe de composition d'objets

Le type t, associé à une classe c reflète partiellement la hiérarchie de composition des objets de cette classe, c'est-à-dire les liens que possèdent ceux-ci avec d'autres objets [ADIB.93], [BENZ.90], [BENZ.93].

Un type t peut être représenté par un graphe orienté étiqueté, dont la définition suit:

Définition 5: **Graphe de type GT (t)**

(i) GT (t) = (N_t, E_t) où:

(ii) N_t est l'ensemble des sommets étiquetés. Chaque sommet représente un type et est étiqueté au moyen de $\beta : N_t \rightarrow T \cup \{$ **tuple** , **set** , **list** $\}$ et $\alpha : N_t \rightarrow C$ où C est l'ensemble des clases.

(iii) Si t est associé à la classe identifié par c alors $t \in N_t$ et $\alpha(t) = c$. Sinon, α (t) n'est pas définie.

(iv) Si t est un atome alors $t \in N_t$ et β (t) = atome.

(v) Si $t_1 \ldots t_n \in N_t$ et t = **tuple** ($a_1 : t_1 , \ldots , a_n : t_n$) alors $t \in N_t$ et $\beta(t)$ = **tuple**.

(vi) Si $t_1 \in N_t$ et t=**set**(t_1) alors $t \in N_t$ et β(t)=**set**.

(vii) Si $t_1 \in N_t$ et t=**list**(t_1) alors $t \in N_t$ et β(t)=**list.**

(viii) E_t est l'ensemble des arcs orientés et étiquetés au moyen de $\gamma : E_t \rightarrow A$ où A est l'ensemble des noms d'attributs.

(xi) Si t = **tuple**(a_1: $t_1, \ldots, a_n : t_n$) alors $(t, t_i) \in E_t$ et γ $(t, t_i) = a_i$ pour tout i de 1 à n.

(x) Si t = **set** (t_1) alors $(t, t_1) \in E_t$ et γ (t, t_1) n'est pas définie.

(xi) Si t = **list** (t_1) alors $(t, t_1) \in E_t$ et γ (t, t_1) n'est pas définie.

La totalité des liens entre tous les types présents dans le système est donnée par le **Graphe des Types GT**.

Définition 6: Graphe des types GT

(i) **GT** = $\cup_{t \in T}$ **GT** (t) = $(\cup_t N_t , \cup_t E_t)$.

La figure 2 illustre cette définition.

La relation d'héritage entre clases induit une relation de sous-typage entre types [LECL.89], [ADIB.93], [BENZ.93]. Dû à la caractérisation syntactique de la relation de sous-typage, si t_1 est un sous-type de t'_1 et s'il existe un arc (t'_1 , t'_2) étiqueté par l dans GT alors il existe un arc (t_1 , t_2) étiqueté par l dans GT avec t_2 sous-type de t'_2.

Définition 7: Graphe de type avec héritage GTH (t)

On ajoute à la définition 5 la ligne suivante:

(xii) Si t_1 est un sous-type de t_2 alors $(t_1 , t_2) \in E_t$ et γ (t_1 , t_2) n'est pas définie. Cet arc en pointillé représente le lien d'héritage.

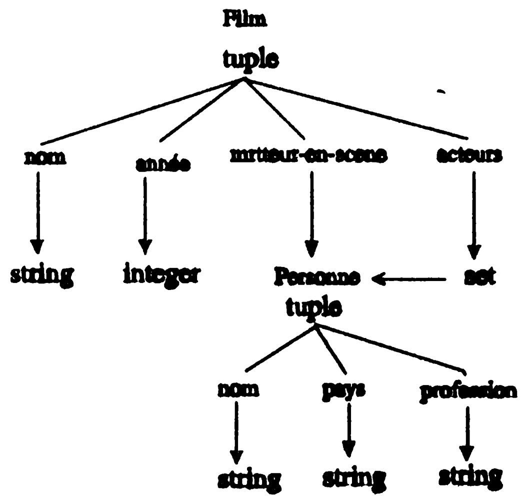

Figure 2. Graphe des types GT.

Définition 8: Graphe des types avec héritage GTH

Le graphe des types avec héritage GTH est défini comme suit:

(i) $\mathbf{GTH} = \bigcup_{t \in T} \mathbf{GTH}(t)$.

Exemple

Soient les définitions suivantes:

add **class** Film-de-Pub **inherit** Film
type **tuple** (metteur-en-scene: Publicitaire)

add **class** Publicitaire **inherit** Personne
type **tuple** (agence: **string**)

add **class** Marin **inherit** Personne
type **tuple** (bateau: **string**)

La figure 3 présente le graphe des types avec héritage.

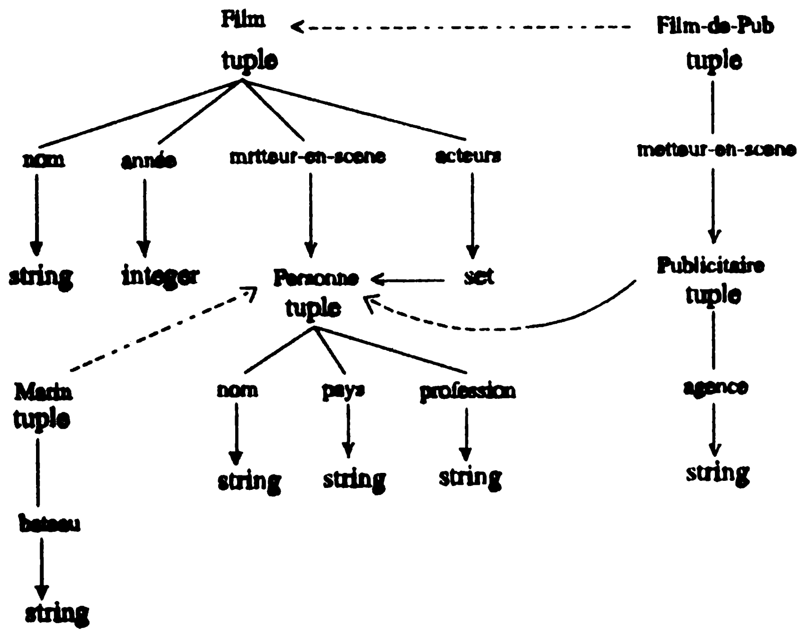

Figure 3. Graphe des types avec héritage.

Etant donné un ensemble d'objets, à nouveau, les liens entre ces objets pourront être représentés par un graphe orienté. Nous appelons ce graphe: **Graphe de Composition d'Objets GCO.**

La définition d'un tel graphe suit:

Définition 9: Graphe de composition d'objets GCO

Soit I un ensemble d'identificateurs d'objets. Le GCO est défini pour les objets/valeurs par:

(i) $G_I = (V_I , E_I)$ où:

(ii) V_I est l'ensemble des nœuds chaque nœud représente une valeur et est étiqueté par $\alpha : V_I \rightarrow$ **I** et $\beta : V_I \rightarrow$**V** $\cup$ {**tuple, set, list**}.

(iii) Si v est associée à l'objet identifié par i alors $v \in V_I$ et $\alpha (v) = i$.

(iv) Si v est une valeur atomique alors $v \in V_I$ et $\beta(v)$=valeur.

(v) Si $v_1...v_n \in V_I$ et v = **tuple** $(a_1 : v_1 , ... , a_n : v_n)$ alors $v \in V_I$ et $\beta(v)$ = **tuple**.

(vi) Si $v_1 ... v_n \in V_I$ et v = **set** $(v_1 ... v_n)$ alors $v \in V_I$ et $\beta (v)$ = **set**.

(vii) Si $v_1...v_n \in V_I$ et v = **list** $(v_1 ... v_n)$ alors $v \in V_I$ et $\beta (v)$ = **list**.

(viii) E_I est l'ensemble des arcs étiquetés au moyen de $\gamma : E_I \rightarrow$ **A**, où **A** est l'ensemble des noms d'attributs.

(ix) Si v = **tuple** $(a_1 : v_1 , ... , a_n : v_n)$ alors

$(v, v_k) \in E_I$ et $\gamma (v, v_k) = a_k$, pour tout k de 1 à n.

(x) Si v = **set** $(v_1 , ... , v_n)$ alors $(v, v_k) \in E_I$

et $\gamma (v, v_k)$ n'est pas définie, pour tout k de 1 à n.

(xi) Si v = **list** $(v_1 , ... , v_n)$ alors $(v, v_k) \in E_I$

et $\gamma (v, v_k)$ n'est pas définie, pour tout k de 1 à n.

Exemple

Soit les objets :

$o_1 = (i_1$, **tuple** (Nom : Ionesco, Conjoint : i_2 , Age : 35 , Enfants : **set** (i_3)))

$o_2 = (i_2$, **tuple** (Nom : Magda , Conjoint : i_1 , Age : 33 , Enfants : **set** (i_3)))

$o_3 = (i_3$, **tuple** (Nom : Chrétien , Conjoint : **nil** , Age: 10, Enfants : **set (nil)**)).

Si O = $\{o_1, o_2, o_3\}$, alors dans la figure 4 nous présentons le graphe **GCO** correspondant.

Le **GCO** peut être vu comme une "instanciation" du graphe des types **GT**.

L'extension d'une classe c est une valeur de type **set** (c), les nœuds correspondant

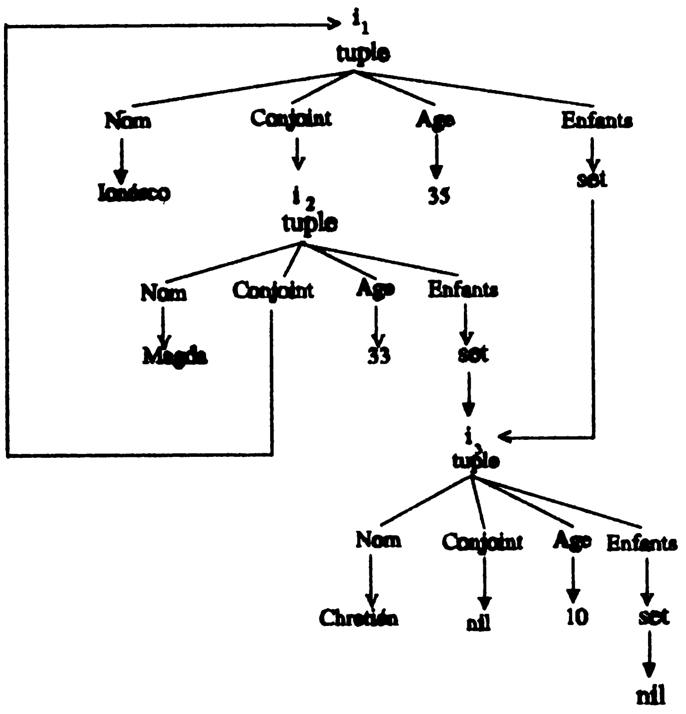

Figure 4. Graphe de composition d'objets

à ces objets seront présents dans le CGO. Le CGO décrit les connexions inter-objet d'un ensemble d'objets donné.

5. Conclusion

Nous avons présenté ici les concepts d'un modèle à objets complexes avec identité d'objet.

Dans les SGBDOO le concept d'objet est fondamental. Nous avons défini le concept d'objet complexe à partir d'objets atomiques et de constructeurs qui s'appliquent récursivement independamment du type des objets.

On représent souvent les liens entre les objets au moyen d'un graphe.

Nous avons construit le graphe des types GT, le graphe des types avec héritage GTH et le graphe de composition d'objets GCO.

Le graphe des types et le graphe de composition d'objets jouent un rôle prépondérant dans la mise en œuvre des stratégies de regroupement d'objets sur disque dans un système de bases de données orienté-objet.

BIBLIOGRAPHIE

[ABIT.87] S. ABITEBOUL, C. BEERI : On the power of languages for the manipulation of complex objects. In Proceedings of theInternational Workshop on Theory and Applications of Nested Relations and Complex Objects, Darmstadt,1987.

[ABIT.88] S. ABITEBOUL, S. GRUMBACH : Col : A logic - based Language for Complex Objects. In Proc. of EDBT International. Conf., 1988.

[ABIT.89] S. ABITEBOUL, P. KANELLAKIS : Object Identity as Query Language Primitive, In Proc. of the ACM SIGACT- SIGMOD Symp.on Principles of Database Systems, juin, 1989.

[ADIB.93] M. ADIBA, C. COLLET : Objets et Bases de Données. Le SGBD O2,Hermès, Paris, 1993.

[ATKI.87] M. ATKINSON, P.BUNEMAN : Types and Persistence in Database Programming Languages, ACM Computing Surveys, June, 1987.

[BANC.86] A. BANCILHON, S. KHOSHAFIAN : A calculus for Complex Objects, ACM PDDS Conference 1986.

[BENZ.89] V. BENZAKEN : Regroupement d'objets sur disque dans un système de bases de données orienté-objet, thèse de doctorat, Paris, 1990.

[BENZ.93] V. BENZAKEN, A. DOUCET : Bases de Données orientées objet. Origines et principes, Armand Colin, 1993.

[CODD.70] E.F.CODD : A Relational Model of Data for Large Shared Data Banks, CACM Vol 13 No 6, June, 1970.

[HUMB.91] M. HUMBERT : Les Bases de Données, Hermès, 1989.

[KHOS.86] S. KHOSHAFIAN, G. COPELAND : Object Identity, OOPSLA 86, Portland Oregon, Sept. 1986.

[LECL.88] C. LECLUSE, P. RICHARD, F. VELEZ : O2, an Object- Oriented Data Model, ACM 3/1988.

[LECL.89] C. LECLUSE, P. RICHARD : Modeling Complex Structures in Object-Oriented Databases, Prooc. of the ACM PODS Conference, Philadelphie, 1989.

[MAKI.77] A. MAKINOUCHI : A Consideration on Normal Form of Not-Necessarly-Normalized Relation in the Relational Model of Data, ACM VLBD Tokio, Japan, 1977.

[SCHE.82] H. SCHEK, G. JAESCHKE : Remarks on the Algebra of Non First Normal Form Relations, ACM PODS Los Angeles, 1982.

[RICH.89] P. RICHARD : Des Objets Complexes aux Bases de Données Orientées - objets, Thèse de doctorat, Paris, 1991.

[ZANI.85] C.ZANIOLO : The Representation and Deductive Retrieval of Complex Objects, VLBD, Stockholm, August, 1985.

STUDIA UNIV. BABEŞ-BOLYAI, MATHEMATICA, XL, 3, 1995

OBJECT-ORIENTED SPECIFICATION IN SOFTWARE DEVELOPMENT

Simona MOTOGNA[*]

Received: 20.11.1995
AMS subject classification: 68N15, 68N05

REZUMAT. - Specificarea obiect-orientată în dezvoltarea software. Specificarea formală joacă un rol important în dezvoltarea de sisteme informatice largi şi complexe. Pe de altă parte, programarea orientată obiect s-a dovedit în ultimii ani ca fiind un instrument cu beneficii clare în dezvoltarea de produse informatice. Scopul acestei studiu este de a propune o metodă de specificaţie orientată obiect bazată pe descompunerea, abstractizarea şi încapsularea sistemului, oferind şi posibilitatea reutilizării.

1. Introduction. We propose an object-oriented specification method which will support widespread reuse and respects the following principles: a specification, in general, must be formal, understandable, as well as abstract and implementation-independent. Reuse of the software components is possible in the same problem or for other similar problems.

The design of a component influences its potential for reuse, but a good design is not always sufficient. The expression of the design is equally important. The specification of a component must be achieved such that the implementation and use of that component meet the following conditions:

- understand easily, but exactly which is the component functionality;
- choose freely among multiple, efficient implementations;
- certify that an implementation satisfies the specification requirements.

[*] *"Babeş-Bolyai" University, Faculty of Mathematics and Computer Science, 3400 Cluj-Napoca, Romania*

2. Specification Features. Object-oriented features can be successfully used to describe abstract entities, and this is exactly what a specification must do - to study the abstract behavior without any constraints about computer architectures. That's why, object oriented concepts like data abstraction and encapsulation (components will be considered classes) will be used in this specification method, as well as inheritance and polymorphism to support software reuse.

The idea of this method is based on Eiffel [4]. The principal reasons for choosing Eiffel are:

- it is an object-oriented language, so it offers data abstraction, encapsulation, inheritance and polymorphism;

- it has a set of assertions which can express the conditions that an operation has to satisfy.

We shall shortly overview which are these assertions, which in this case will be specified in conditions.

- Preconditions will be specified through a **requires** clause and represent the conditions under which the operation will function correctly.

- Postconditions will be specified through an **ensures** clause and represent the conditions assured after performing the corresponding operation.

If the precondition of an operation holds before an operation is invoked, then the postconditions will be guaranteed to hold when the operation completes, assuming a correct implementation of the specification.

In addition we suppose that each parameter of an operation has one of the following modes: *conserves, uses, produces* or *modifies*:

- the *conserves* mode indicates that the parameter value will remain unchanged during the operation performing (like the invariant assertion in Eiffel);

- the *uses* mode indicates that a parameter value is used by the operation and that the initial value is not modified;

- the *produces* mode indicates that the obtained value is relevant or that the parameters has a value only after performing the operation;

- the *modifies* mode is used when a parameter has an input value that is modified and returned by an operation.

These modes are only specification notations and should not be confused with parameter passing mechanisms. They are included in the specification only to increase the understandability of it.

The interface of a class describes what the component provides and these are the only data and operations which are visible to other components.

The reuse of the component in defining other components of the same system is achieved using inheritance, specified by an *inherits* clause. Reusing this component in another system is easy to achieve since the specification of a component is encapsulated.

In order to understand this method of specification, the following example will be considered:

Specification of a generic **Stack**

```
class Stack[Item]
Type content: sequence of Item

interface
 init, empty, push, pop, top
end;

operation init
        parameters:{produces c:content
                   }
        ensures c = []
end_operation

operation empty
```

```
        parameters:{conserves c:content
                    produces b:Bool
                    }
        ensures (b = c = [])
end_operation

operation push
        parameters:{modifies c:content
                     uses i:Item
                    }
        ensures ( not empty ) ∧ (c=c+[i])
end_operation

operation pop

        parameters:{modifies c:content
                     produces i:Item
                    }
        requires (not empty)
        ensures (c=c-[i])
end_operation

operation top
        parameters:{conserves c:content
                     produces i:Item
                    }
        requires (not empty)
        ensures (not empty)
end_operation

end_class --class Stack
```

This example shows a way of specifying stacks regardless of the elements type. A stack component must provide operations as creation (init), a test: is the stack empty? (empty), the classical operations push, pop and top. Instead of defining these operations as procedures or functions, the parameters and their mode are specified separately, between braces. Then, the preconditions and postconditions are described.

In general, a specification of a component will be:

```
class <class_name>
inherits <class_name>                    // the class from each
                                                //inherits
Type ...                                        // user defined types
interface                                // operations exported
 <op1>,<op2>,...
end;

operation <name_op>
       parameters:{[conserves <var>:<type>,..]
                         [uses <var>:<type>,..]
                         [produces <var>:<type>,..]
                         [modifies <var>:<type>,..]
                         }
       [requires (cond1) [∧ (cond2) ...]]
       [ensures (cond1) [∧ (cond2) ...]]
end_operation
...                                             // the same format for each
                                                // operation
end_class
```

The complete syntactic definition of the language is given in Appendix A.

It's easy to observe that this specification method is not intended to suggest any implementation method. The purpose of specification is, on the contrary, to give more choices of implementation.

3. Conclusions. The specification method described above meets the criteria regarding formal specification, and also offers flexibility and security. The idea of this specification was found in [2], but lacks in information regarding the operations specification. The model proposed has added some techniques in order to give a more clear specification for each operation included in a class.

The purpose of it is to fit between the object-oriented analysis and design and an object-oriented implementation.

This study represents more an idea which can be applied with fruitful results, especially in reusing software components.

The specification method is implementation independent. Any object-oriented programming language can be used for implementation, but this is not a requirement: this kind of specification can be used for any other language without object oriented features, transforming these class definitions in user defined types and the operations in procedures and functions. The dezadvantage of such a language is that properties like inheritance, polymorphism and encapsulation are not available and the programmer has to find a way to "translate" the given specification using the language facilities.

Appendix A: Syntactic Definition

The syntactic definition is given in extended BNF (Backus-Naur Form). The following notational conventions are used:

- the keywords are bold;
- <> item enclosed in angle brackets are required
- [] item enclosed in square brackets are optional
- { } items enclosed in braces may be repeated zero or more times
- // everything that follows up to the end of line denotes comment

```
<class_description> ::= class <class_name> [[<formal_type_parameters>]]
                        <inheritance_declaration>
                        Type <type_definition>
                        <interface_declaration>
                        <operation_description>
                        end_class

<class_name> ::= <identifier>
<formal_type_parameters> ::= <identifier_list>
<identifier_list> ::= <identifier> {, <identifier>}

<inheritance_declaration> ::= inherits <class_list>
<class_list> ::= <class_name> {, <class_name>}

<type_definition> ::= <identifier> : <type>
```

```
<type> ::= Integer | Bool | Sequence of <identifier> |...*

<interface_declaration> ::=   interface <op_name> {, <op_name>}
                              end
<op_name> ::= <identifier>

<operation_description> ::= <operation_item> {, <operation_item>}
<operation_item> ::=    operation <op_name>
                        parameters: {[conserves <var_decl_list>]
                                     [uses <var_decl_list>]
                                     [produces <var_decl_list>]
                                     [modifies <var_decl_list>]
                                   }
                        [requires <condition_list>]
                        [ensures <condition_list>]
                        end_operation
<var_decl_list> ::= <var_list> : <type>
<var_list> ::= <var_name> {, <var_name>}
<var_name> ::= <identifier>
<condition_list> ::= <condition> { ∧ <condition>}
<condition> ::= <bool_exp> | not <bool_exp> |
                <bool_exp> V <bool_exp> |<bool_exp> ∧ <bool_exp>
<bool_exp> ::= <exp> <rel_operator> <exp> | <op_name>**
<rel_operator> ::= < | > | = | <> | <= | >=
<exp> ::= <var_name> | <constant> | <exp> <operator> <exp>
<operator> ::= + | - | * | /
<constant> ::= <number>
<number> ::= <digit>{<digit>}
<digit> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0
```

* It can be completed with other types, when it is necessary

** Note that an operation which returns a boolean value can be considered an boolean expression

REFERENCES

[1] L. Cardelli, P. Wegner - On understanding types, data abstraction and polymorphism, Computing Surveys, 17(4), pp.471-522.

[2] Y Cheon, G. T. Leavens - The Larch/Smalltalk Interface Specification Language, ACM Transactions on Software Engineering and Methodology, July 1994, vol. 3 no.3, pp. 221-253.

[3] B. Meyer - Object-Oriented Software Development, Prentice-Hall, 1988

[4] P. Wegner - Concepts and Paradigms of Object-Oriented Programming, OOPSLA'89 Keynote Talk, OOPS Messenger, vol.1,no.1, pp. 7-87, 1990

/

STUDIA UNIV. BABEŞ-BOLYAI, MATHEMATICA, XL, 3, 1995

FUZZY DECISION SUPERVIZED CLASSIFIERS

Horia F. POP*

Received: February 1, 1995
AMS subject classification: 68P99

Rezumat. Clasificatori supervizaţi cu decizie nuanţată. In acest articol sunt prezentate o serie de generalizări ale unor algoritmi de instruire clasici. Sunt prezentate câteva din problemele generate de aceştia. Apoi se construieşte un algoritm de clasificare supervizată nuanţată bazat pe o generalizare a algoritmului Fuzzy n-Medii. Sunt studiate proprietăţile sale şi sunt trecute în revistă câteva avantajele obţinute prin utilizarea sa.

1. Introduction

Let us consider a set of objects, $X = \{x^1, ..., x^p\} \subset \mathbf{R}^d$, classified with a fuzzy clustering algorithm of the type Fuzzy n-Means, and the fuzzy partition $P = \{A_1, ..., A_n\}$ coresponding to the cluster substructure of the set X (see [2,3]).

We rise the problem of including an extra-object $x^0 \notin X$ in the cluster structure of X. Of course, this would mean to determine the membership degrees of x^0 to the fuzzy sets members of the partition P. These degrees will provide sufficient information in order to classify the object x^0 with respect to the elements of X.

The algorithms that solve this kind of problems we be called *fuzzy decision supervized classification algorithms*. Supervised classification because the classification of the extra-object is realized using not only the data set X, but also the fuzzy partition obtained by classifying the set X. Fuzzy decision because, unlike the traditional classifiers, where the aim is to state in which classical subset the object may be included, now we are interested in the membership degrees of the object to the fuzzy sets members in the given fuzzy partition.

* *"Babeş-Bolyai" University, Faculty of Mathematics and Computer Science, 3400 Cluj-Napoca, Romania*

The simplest approach is the classification of the extended set $X \cup \{x^0\}$ using one of the common fuzzy clustering algorithms (see, for instance, [3]), and the comparation of the produced partition to the partition P. Although, this method is very costly considering the neccessary execution time, because it supposes the classification of the objects of X, set that in the real applications may be quite large. Also, this is not supervised classification, because the information provided by the fuzzy partition P is not used.

The alternative approach is to keep unchanged the membership degrees of the objects in X to the sets of the fuzzy partiton P, and to determine the membership degreess of x^0 as a consequence of the minimization of an objective function similar to those used for the algorithms of the type Fuzzy n-Means.

Also, we will present in this paper some more straightforward algorithms of this type. These algorithms are fuzzy generalizations of the well-known **k nearest neighbours and nearest prototype**.

2. The algorithm of the k nearest neighbours

The algorithm of the k nearest neighbours is one of the standard methods of the supervised classification and it has been especially remarked because of its simplicity. The method is based on the evaluation of the memberships of an unknown object to the classes of the given set using the distances between this object and its k nearest neighbours, and the memberships of these neighbours. The method presented here is a development of the simpler variant that attributes an object to the class that contains its nearest neighbour.

Let us consider a set of classified objects, $X = \{x^1, ..., x^p\} \subset \mathbf{R}^d$ and the fuzzy partition $P = \{A_1, ..., A_n\}$ corresponding to the cluster substructure of the set X. Let us consider $x^0 \in \mathbf{R}^d$ an object that needs to be classified with respect to the fuzzy partition P.
The classification of the object x^0 will be realized by examining a number of k objects from X which are the nearest ones to x^0 (the most similar).

Even if by this method will be produced n new fuzzy sets on $X \cup \{x^0\}$, denoted $\tilde{A}_i$ and having the property $\tilde{A}_i(x^j) = A_i(x^j)$ for every $i=1,...,n$ and $j=1,...,p$, the new fuzzy sets will also be denoted by A_i.

In what follows we will consider a dissimilarity measure on $X \cup \{x^0\}$ (see [3,5]). Let d be a metric in the $\mathbf{R}^d$ space.

The *dissimilarity* between the object x^0 and a certain object x^j of the set X may be defined as

$$D(x^0, x^j) = d^2(x^0, x^j),\ j=1,...,p \tag{1}$$

and may be interpreted as a measure of the "non-ressemblence" between the objects x^0 and x^j.

Let σ be a permutation of the set $\{1,...,p\}$, so that

$$D(x^0, x^{\sigma(i)}) \leq D(x^0, x^{\sigma(j)}) \leftrightarrow i \leq j. \tag{2}$$

In order to determine the membership degrees of the object x^0 we will need to take into consideration with differnt weights the membership degrees of the k nearest neighbours ($x^{\sigma(j)}$, $j \leq k$). The more similar $x^{\sigma(j)}$ is to x^0, the greater weight its memberships should be given. This remark leads us to the following empirical rule for computing the memberships of x^0:

$$A_i(x^0)=\frac{\sum_{j=1}^{k}\frac{A_i(x^{\sigma(j)})}{D(x^0,x^{\sigma(j)})}}{\sum_{j=1}^{k}\frac{1}{D(x^0,x^{\sigma(j)})}} \tag{3}$$

The obtained algorithm is called **the algorithm of the k nearest neighbours**:

S1 Be given X, P, k.

S2 Computes the dissimilarities with respect to the relation (1).

S3 Determines the permutation σ such that the condition (2) be verified.

S4 Computes the membership degrees of the object x^0 with respect to the relation (3).

Remark. The membership degrees computed in this way verify the relation

$$\sum_{i=1}^{n} A_i(x^0)=1.$$

Remark. An alternative to this method is not to take into account all the objects in X, but only a subset of them, namely the most representative ones, eventually those having the membership degree to one of the classes at least 0.8.

3. The algorithm of the nearest prototype

Let us consider a set of classified objects, $X = \{x^1,\dots,x^p\} \subset \mathbf{R}^d$ and the fuzzy partition $P = \{A_1,\dots, A_n\}$ corresponding to the cluster substructure of the set X. Let $x^0 \in \mathbf{R}^d$ be an object that needs to be classified with respect to the fuzzy partition P.

In what follows we will suppose that the clusters of the set X have a hyperspherical shape. Moreover, we will suppose that in order to identify the optimal fuzzy partition P, the Fuzzy n-Means algorithm has been used. We consider the clusters as being represented by puctual prototypes. We will denote the prototype of the class A_i with L^i, $L^i \in \mathbf{R}^d$. As stated by the Fuzzy n-Means algorithm, the expression of the prototypes L^i is given by

$$L^i=\frac{\sum_{j=1}^{p}(A_i(x^j))^2x^j}{\sum_{j=1}^{p}(A_i(x^j))^2}. \tag{4}$$

The unknown object x^0 will be classified with respect to the distance between it and the prototypes L^i of the classes members of the partition P. We will consider that the membership degree of the object x^0 to a certain class is as larger as the object is nearer the prototype of that class.

In this case we will also use the same notation A_i to denote the extended fuzzy sets, defined over $X \cup \{x^0\}$.

In what follows we will consider a dissimilarity measure over $X \cup \{x^0\}$. Let d be a metric in the $\mathbf{R}^d$ space.

The ***dissimilarity*** between the object x^0 and the prototype L^i of the set A_i is defined as

$$D_i(x^0, L^i) = (d_i(x^0, L^i))^2,$$

where d_i is the local metric induced by the metric d and by the fuzzy set A_i. The dissimilarity D may be interpreted as a measure of "nonresemblance" between the object x^0 and the prototype L^i. Using the definition (see [5]), it may be written as:

$$D_i\}(x^0, L^i) = (A_i(x^0))^2\, d^2(x^0, L^i). \tag{5}$$

The inadequacy between the memberships $A_i(x^0)$ of the object x^0 to the n classes and the prototypes L^i of these classes may be written using the function $J(A_1(x^0),...,A_n(x^0))$ given by

$$\begin{aligned} J&=\sum_{i=1}^{n} D_i(x^0,L^i)\\ &=\sum_{i=1}^{n} (A_i(x^0))^2 d^2(x^0,L^i). \end{aligned} \tag{6}$$

So our problem may be reduced to the determination of those membership degrees $A_i(x^0)$ which minimize the objective function J. This result is stated by the following

Theorem. The membership degrees $A_i(x^0)$, $i=1,...,n$, are a minimum of the function J if and only if

$$A_i(x^0)=\frac{1}{\sum_{k=1}^{n}\frac{d^2(x^0,L^i)}{d^2(x^0,L^k)}} \tag{7}$$

The proof of this theorem, as the proofs of all the theorems in this paper, are very similar to the proofs of the minimality theorems presented in [2,3]. For this reason we will not give here any explicit proof.

The algorithm obtained using this theorem will be called **the algorithm of the nearest prototype**:

S1 Be given X and P.
S2 Determine the prototypes L^i using the relation (4).
S3 Computes the distances $d(x^0, L^i)$ from the object x^0 to the prototype L^i of the class A_i.
S4 Computes the membership degrees of the object x^0 with respect to the relation (7).

Remark. The membership degrees $A_i(x^0)$ computed using the relation (7) verify the initial supposition: the smaller the distance from x^0 to a prptptype is, the greater the membership degree of x^0 to that class will be.

Remark. The computational effort is more reduced at this method as compared to the previous one, if we take into account the fact that generally the number of classes is very much smaller than the number of object: it is enough to compute n distances as compared to p distances for the previous case.

Remark. The geometrical locus of the points x^0 characterized by equal memberships to the classes A_i and A_j, $i,j=1,\dots,n$, $i\neq j$, is the median hyperplane of the segment L^iL^j (which contains the points equally distanced from L^i and L^j). As a consequence, this algorithm presents the problem of the inequal clusters, considering that a point x0 from the outer part of a greater class has many chances to be captured by a smaller neighbour class.

Remark. In order to overpass the problem of inequal clusters, when producing the fuzzy partition P the adaptive version of the Fuzzy n-Means algorithm may be used (see [2]). The local adaptive distance defined for that algorithm may also be used to determine the membership degrees $A_i(x^0)$.

4. The Restricted Fuzzy n-Means algorithm

Let us consider a set of classified objects, $X=\{x^1,...,x^p\} \subset \mathbf{R}^d$ and the fuzzy partition $P=\{A_1,...,A_n\}$ corresponding to the cluster substructure of the saet X. Let $x^0 \in \mathbf{R}^d$ be an object that needs to be classified with respect to the fuzzy partition P.

Let us suppose that the partition P has been produced using the Fuzzy n-Means algorithm. Our aim is to develop an algorithm that should compute the optimal fuzzy partition $\tilde{P}$ corresponding to the set $\tilde{X} = X \cup \{x^0\}$, by using a mechanism of the type of Fuzzy n-Means, with the difference that the membership degrees of the objects in X to the classes A_i, $i=1,...,n$ may not be modified.

In what follows we will consider a metric d in the Euclidean space $\mathbf{R}^d$. We will suppose that d is norm induced, so
$d(x, y) = (x-y)^T M (x-y)$, $\forall\, x, y \in \mathbf{R}^d$, where M is a symmetrical and positively defined matrix.

Let us remember that the objective function of the Fuzzy n-Means algorithm is

$$J(P,L)=\sum_{i=1}^{n}\sum_{j=1}^{p}(A_i(x^j))^2 d^2(x^j,L^i), \tag{8}$$

where L^i is the pointly prototype associated to the fuzzy class A_i.

So, the objective function we will have in mind in this case is

$$\tilde{J}(\tilde{P},L)=\sum_{i=1}^{n}\sum_{j=0}^{p}(A_i(x^j))^2 d^2(x^j,L^i), \tag{9}$$

with the mention that $A_i(x^j)$ are kept constant for each i and for $j=1,...,p$.

The classification problem reduces to the determination of the fuzzy partition $\tilde{P}$ and of the representation L that minimizes the function $\tilde{J}$. The main result with this respect is given by the following

Theorem. (i) The fuzzy partition $\tilde{P}=\{A_1,...,A_n\}$ is minimum of the function $\tilde{J}(.,L)$ if and only if

$$A_i(x^0)=\frac{1}{\sum_{k=1}^{n}\frac{d^2(x^0,L^i)}{d^2(x^0,L^k)}} \tag{10}$$

(ii) The set of prototypes $L=\{L^1,...,L^n\}$ is minimum of the function $\tilde{J}(\tilde{P},.)$ if and only if

$$L^i=\frac{\sum_{j=0}^{p}(A_i(x^j))^2x^j}{\sum_{j=0}^{p}(A_i(x^j))^2}. \tag{11}$$

With this result, the optimal membership degrees of x^0 to the classes A_i will be determined using an iterative method in which $\tilde{J}$ is succesively minimized with respect to $\tilde{P}$ and L. The process will start with the initialization of the prototypes L^i to the values corresponding to the optimal positions computed for the function J. The resulted algorithm, the **Restrictive Fuzzy n-Means Algorithm**, is the following:

S1 Be given X and P.

S2 Determine the initial positions of the prototypes L^i as the optimal positions computed for the function J.

S3 Determine the membership degrees $A_i(x^0)$, $i=1,...,n$, using the relation (10).

S4 Determine the new positions of the prototypes L^i using the relation (11).

S5 If the new prototypes are closed enough to the former ones, then stop, else go back to the step S3.

In what follows we propose to determine the geometrical locus of the points x^0 for which the membership degrees to two classes A_{i1} and A_{i2} are equal. Let us denote

$$A_{i1}(x^0) = A_{i2}(x^0) = a. \qquad (12)$$

Firstly, let us denote by L^{*i}, $i=1,\dots,n$ the protptypes corresponding to the fuzzy partition P over X, as they have been computed using the Fuzzy n-Means Algorithm. Thus,

$$L^{*i} = \frac{\sum_{j=1}^{p} (A_i(x^j))^2 x^j}{\sum_{j=1}^{p} (A_i(x^j))^2}. \qquad (13)$$

Let us denote by α_i^2 the following value:

$$\alpha_i^2 = \sum_{j=1}^{p} (A_i(x^j))^2.$$

Then, between the prototypes L^i and L^{*i} exists the relationship

$$x^0 - L^i = (x^0 - L^{*i}) \cdot \frac{\alpha_i^2}{\alpha_i^2 + (A_i(x0))^2}. \qquad (14)$$

The relationship above may be written under the form

$$d(x^0, L^i) = d(x^0, L^{*i}) \cdot \frac{\alpha_i^2}{\alpha_i^2 + (A_i(x0))^2}, \qquad (15)$$

or

$$d(x^0, L^{*i}) = d(x^0, L^i) \left(1 + \frac{(A_i(x^0))^2}{\alpha_i^2} \right). \qquad (16)$$

The relation (10) which computes the membership degrees may be rewritten as follows:

$$A_i(x^0) = \frac{\frac{1}{d^2(x^0, L^i)}}{\sum_{k=1}^{n} \frac{1}{d^2(x^0, L^k)}} \qquad (17)$$

(ii) The set of prototypes L={

From the relations (12) and (17) it results that

$$d(x^0, L^{i1}) = d(x^0, L^{i2}).$$

By relating to (16), it results

$$\frac{d(x^0,L^{*i_1})}{d(x^0,L^{*i_2})}=\frac{1+\frac{a}{\alpha_{i_1}^2}}{1+\frac{a}{\alpha_{i_1}^2}}=K, \tag{18}$$

where

$$K\begin{cases}<1 & if\ \alpha_{i_1}^2>\alpha_{i_2}^2\\ =1 & if\ \alpha_{i_1}^2=\alpha_{i_2}^2\\ >1 & if\ \alpha_{i_1}^2<\alpha_{i_2}^2\end{cases} \tag{19}$$

So, we have obtained the following

Theorem. The geometrical locus of the points x^0 having equal memberships to the classes A_{i1} and A_{i2} is on a hypersphere with the center in the point

$$c=L^{*i_2}\frac{K^2}{K^2-1}-L^{*i_1}\frac{1}{K^2-1}$$

and with the radius

$$r=d(L^{*i_1},L^{*i_2})\frac{K}{|K^2-1|},$$

where K is that stated in the relation (19).

At a more careful analysis of this hypersphere's equation, we may do some interesting remarks.

Remark. This hypersphere "catches" inside itself the prototype

L^{*i} of the class having the largest index α_i^2.

Remark. This may be the main problem of the method and it indicates a dependency with respect to the dimensions of the classes. Although the effect seems to be even more important than to the traditional Fuzzy n-Means algorithm, the experiments done with this algorithm on test examples show that the effects have simmilar dimensions.

Remark. The problem may be solutioned in the same way the problem of the inequal sized clusters (see [2]) has been solved for the Fuzzy n-Means algorithm, namely by using an adaptive metric, with respect to which all the clusters having equal dimensions.

Remark. The center of the hypersphere is always outside the segment L^{*i1} L^{*i2}.

Remark. The hypersphere intersects the segment L^{*i1} L^{*i2} in the point

$$v = L^{*i_1}\frac{1}{K+1} + L^{*i_2}\frac{K}{K+1}$$

which is the more distanced by the center of the segment the more distanced by 1 is K.

Remark. The hypersphere is entirely situated on one side of the median hyperplane of the segment L^{*i1} L^{*i2}.

Remark. Moreover, for $K = 1$, the hypersphere degenerates to the median hyperplane specified above.

R E F E R E N C E S

1. James C. Bezdek. Pattern Recognition with Fuzzy Objective Function Algorithms. Plenum Press, New York, 1981.

2. D. Dumitrescu. Hierarchical pattern classification. Fuzzy Sets and Systems 28 (1988), 145--162.

3. D. Dumitrescu. Classification Theory. Univ. of Cluj-Napoca Press, 1991.

D. Dumitrescu, Horia F. Pop. Convex decomposition of fuzzy partitions, I,II. Fuzzy Sets and Systems (1995). to appear.

Lotfi A. Zadeh. Fuzzy sets. Information and Control 8 (1965), 338--353.

STUDIA UNIV. BABEŞ-BOLYAI, MATHEMATICA, XL, 3, 1995

ON MEDIAN FOR ONE SPECIAL SPACE

Petru SOLTAN and Chiril PRISĂCARU*

Received: 30.10.1995
AMS subject classification: 52,90C

REZUMAT. - Asupra medianei unui spaţiu special. În lucrare se formulează următoarea problemă: fie $\mathbb{R}^2$ planul vectorial definit peste câmpul numerelor reale cu norma $\|x\| = |x^1| + |x^2|$. Considerăm poligonul $M \subset \mathbb{R}^2$, topologic echivalent cu cercul euclidian şi ale cărui laturi sunt paralele cu una din axele sistemului de coordonate din $\mathbb{R}^2$ şi mulţimea de puncte $S = \{x_1, x_2, ..., x_m\} \subset M$, care are ponderile $p(x_1), ..., p(x_m)$ pozitive. În acest articol ne propunem să formulăm un algoritm, bazat pe d-convexitate, pentru un punct $x_0 \in M$ care minimizează funcţia $f(x) = \sum_{x \in M} p(x_i)d(x,x_i)$.

The following problem is formulated in [1]. Let R be a vector plan on the field of real numbers with norm $\|x\| = |x^1| + |x^2|$, $M \subset R^2$ is a poligon, egual topologically with an euclidian circle, andevery side of it is parallel (see figure) to one axis of coordinate system of R^2, $S = \{x_1, x_2, ..., x_m\} \subset M$ be a set of points having respectively the positive weights $p(x_1), p(x_2), ..., p(x_m)$.

It is required to find a median in M, i.e. such a point $x_0 \in M$ that minimizes the function

$$f(x) = \sum_{x \in M} p(x_i)\, d(x, x_i),$$

where $d(x,x_i)$ represents a distance between points x and x_i calculated following a curve of a minimal length in space $M \subset R^2$ that connects these points.

This problem is solved in [1] using complicated algorithm having however advantage of linear complexity.

In this paper the other algorithm for indicated problem is offtered; it is based on d-

*State University of Moldau, Faculty of Mathematics and Cibernetics, Chişinău, Republic of Moldau convexity theory and follows from algorithms developped for finding a median in [2]. The maximal parallel to axes segments (by inclusion in *M*) through any point of local unconvexity [4] (see point x_m on the figure) of a polygon *M* and set a *S* are drawn. So, a polygon *M* is transformed to figure that is divided into parallelograms with sides forming a grid in *M*; the last is denoted by graph $G = (X, V)$, where *X* is the set of vertices (nodes), and *V* is the set of edges. The points $x_1, x_2, \ldots, x_m$ represent o subset of vertices of *G*. Denote the other vertices of *G* by $x_{m+1}, x_{m+2}, \ldots, x_n$. Hence, we obtain the new set of points *X* of a polygon *M*.

Now we assign a new positive weight to every point $x \in X$. If $x_i \in X \setminus S$, then we to x_i the weight $q(x_i) = 1$, $(i = m+1, \ldots, n)$, if $x_i \in S$ then $q(x_i) = p(x_i) + 1$, $(i = 1, \ldots, m)$. The new problem is formulated in the following way: to find such a vertex $x_0 \in X$ of graph *G* that minimizes the function

$$\varphi(x) = \sum_{x \in X} q(x_i)\, d(x, x_i).$$

According to norm $\|x\|$ in R^2 and to results of [2], $d(x,x_i)$ of this new problem is the same distance that participates in definition of the function $f(x)$.

Applying the reasoning that was presented in [2], we obtain that graph *G* satisfied the conditions indicated in this paper which permit to solve a new formulated problem by the same algorithm, since it is not necessary to know distances $d(x,x_i)$, $i = 1, 2, \ldots, n$.

The essence of this algorithm is the following.

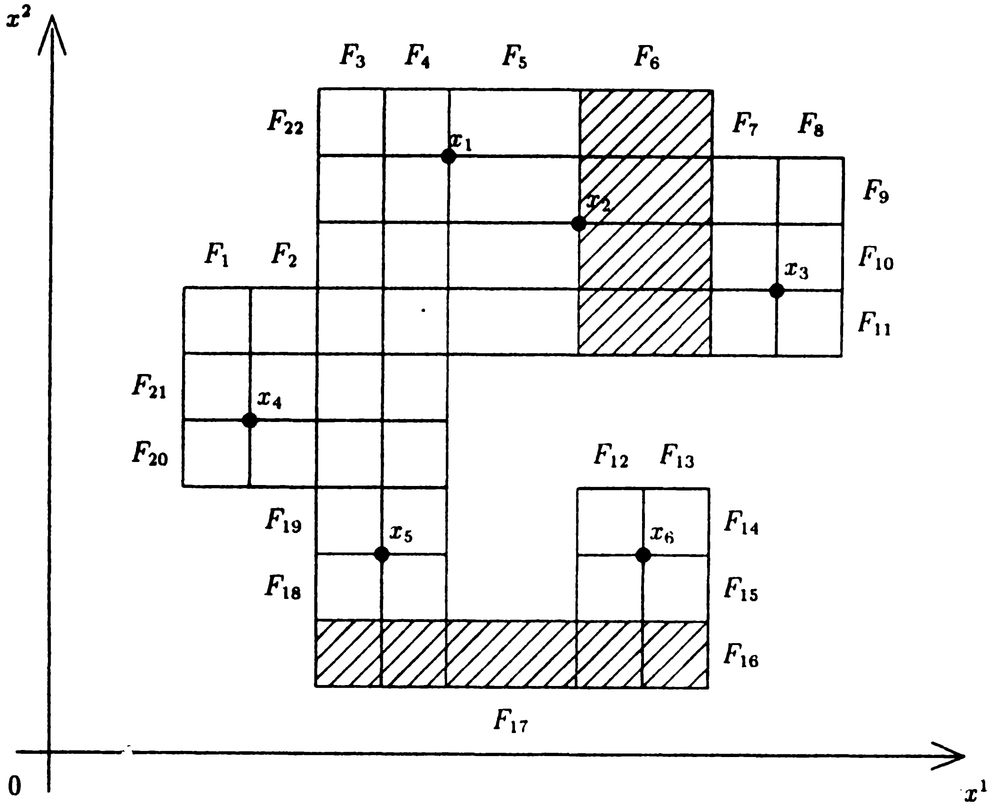

Fig. 1

Let s be the number of horisontal and vertical strips in which polygon M is devided. Denote these strips by F_1, F_2, ..., F_j, ..., F (on the figure the examples of strips are indicated by horisontal sides and $s = 22$).

Put in correspondence to any vertex of graph G a sequence from 0 and 1 of length s according to the following conditions: 1) to point x_1 it is put in correspondence sequence $\varepsilon^1 = (\varepsilon_1^1, \varepsilon_2^1, \dots, \varepsilon_j^1, \dots, \varepsilon_s^1)$, where $\varepsilon_j^1 = 0$, $j = 1, 2, \dots, s$; 2) to point x_i, $i = 1, 2, \dots, n$ it is put a sequence $\varepsilon^i = (\varepsilon^i_1, \varepsilon^i_2, \ldots, \varepsilon^i_j, \ldots, \varepsilon^i_s)$, where $\varepsilon^i_j = 0$, if an arbitrary chain of a graph G connecting the vertices x_1 and x_i intersects a piece F_j even number of time, and $\varepsilon^i_j = 1$, otherwise. So, we construct a function $\alpha : X \to \{\varepsilon^1, \varepsilon^2, \ldots, \varepsilon^i, \ldots, \varepsilon^n\}$, which as it follows from [2] is one-to-one.

Further construct a new sequence $r = (r^1, r^2, \ldots, r^j, \ldots, r^s)$ of elements 0 and 1 in correspondence to the conditions:

1) $r_j = 0$, if

$$\sum_{i=1}^{n} q(x_i)(1 - \varepsilon^i_j) > \frac{1}{2}\sum_{i=1}^{n} q(x_i),$$

2) $r_j = 1$, if

$$\sum_{i=1}^{n} q(x_i)(1 - \varepsilon^i_j) < \frac{1}{2}\sum_{i=1}^{n} q(x_i),$$

3) $r_j = 0$ or $r_j = 1$, if

$$\sum_{i=1}^{n} q(x_i)(1 - \varepsilon^i_j) = \frac{1}{2}\sum_{i=1}^{n} q(x_i),$$

If virtue of [2] there exists an index $i_0, 1 \le i_0 \le n$ such that $r = \varepsilon^{i_0}$. If x_0 is such a vertex of a graph G for which we have $\alpha(x_0) = \varepsilon^0 = r$, then $x_0 = \alpha_{-1}\varepsilon^0$ minimizes the function $\varphi(x)$. Moreover the following statement holds:

THEOREM 1. *The vertices of graph $G = (X, U)$ that minimize the function $\varphi(x)$ are the vertices which give the minimal values for the $f(x)$.*

Proof. Consider the function

$$\varphi_\varepsilon(x) = \sum_{i=1}^{n} q^\varepsilon(x_i)\, d(x, x_i),$$

where ε is an arbitrary positive number, and $q^\varepsilon(x_i) = p(x_i)$ if $x_i \in S$ and $q^\varepsilon(x_i) = p(x_i) + \varepsilon$ if $x_i \in X \setminus S$.

It is obvious that the function $\varphi(x)$ satisfies conditions 1)-3) if and only if the function

$\varphi_\varepsilon(x)$ also satisfies these conditions. Therefore by virtue of [2] the vertices of a graph $G = (X,U)$, that minimize functions $\varphi(x)$, $\varphi_\varepsilon(x)$, are the same.

Nowe extend the function $\varphi_\varepsilon(x)$ that is defined on the set X, onto the wholepoligon $M \subset R^2$ preserving the same notation for it. Evidently, it is easy to do this operation by virtue of definitions for distance $d(x,x_i)$ between points x,x_i and norm $\|x\|$ of the space R^2. By the same way transform the function $f(x)$ by

$$f(x) = \sum_{i=1}^{n} p(x_i)\, d(x, x_i),$$

where $p(x_i) = 0$ for $i = m+1, m+2, ..., n$, that is for $x \in X \backslash S$. Observe that for any $x \in M$ we have the relations $\varphi_\varepsilon(x) - f(x) = n \cdot \varepsilon > 0$. Hence if $\varepsilon \to 0$ for any $x \in M$ we obtain $(\varphi_\varepsilon(x) - f(x)) \to 0$. It follows from this that point $x_0 \in M$ minimising the function $\varphi_\varepsilon(x)$ will minimize the function $f(x)$ (the inverse statement generally does not hold). The theorem is proved.

This theorem permits to reduce the solving of the initial problem to finding the median for function $\varphi_\varepsilon(x)$. So, we obtain that in case $\varepsilon = 1$ algorithm of finding the median for the function $\varphi(x)$ is the same for finding the median for the function $f(x)$.

Note. As it is proved in [3], the set of arguments minimizing the function $\varphi(x)$, is d-convexe. Hence applying in addition the results of [2], we oftain that the set of medians of the function $\varphi(x)$ may represent one of the following possibilities: a) case when we have asingle sequence r, respectively, one point $x_0 = \alpha^{-1}(r)$, 2) case when we have two sequences r_1 and r_2, respectively, one segment $u = [x^1 = \alpha^{-1}(r_1), x^2 = \alpha^{-1}(r_2)] \subset U$ that is parallel to one of the axes; 3) case when we have four sequences r_1, r_2, r_3, r_4, we obtain respectively one paralelogram that divides poligonul $M \subset R^2$ having vertices $x^1 = \alpha^{-1}(r_1), x^2 = \alpha^{-1}(r_2), x^3 = \alpha^{-1}(r_3), x^4 = \alpha^{-1}(r_4)$.

One can prove that this property remains valide for the function $f(x)$, but in this case, if M_φ and M_f are the sets of respective median for $\varphi, f(x)$ then $M_\varphi \subset M_f$.

Direct realization of expounded method by one algorithm gives the possibility to obtain the complexity $o(n^2)$, where n is the number of strips, and it is equal to the sum compiled from the number of given points m and the number of the edges of a polygon k. This complexity is determined by the mode of representation of a grid obtained as a result of polygon's division.

Indeed, the further calculations may be reduced to finding the median of two trees: one that is determined by the horisontal strips and the other determined by the vertical strips. To every strips corresponds an edge of a tree. Two edges have one common vertex if the respective strips have the common border. The weigts of this vertex equals to the sum of weights of vertices on grid that belong to this border.

For example for polygon pictured on fig.1 the horizontal tree is H_o, the vertical tree is H_v (fig. 2).

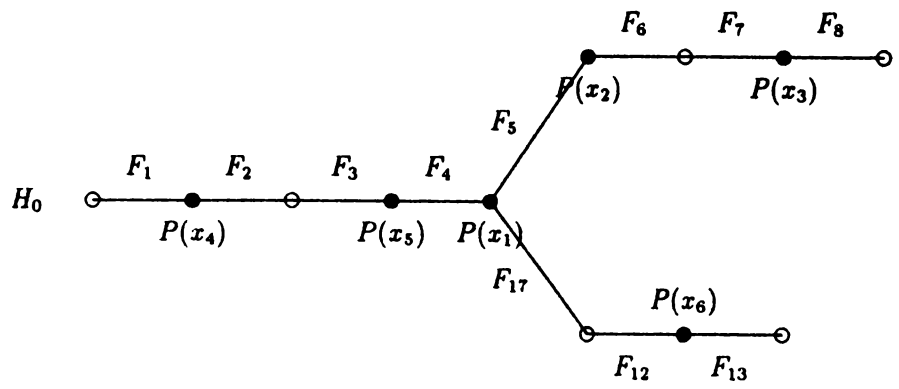

Fig. 2a

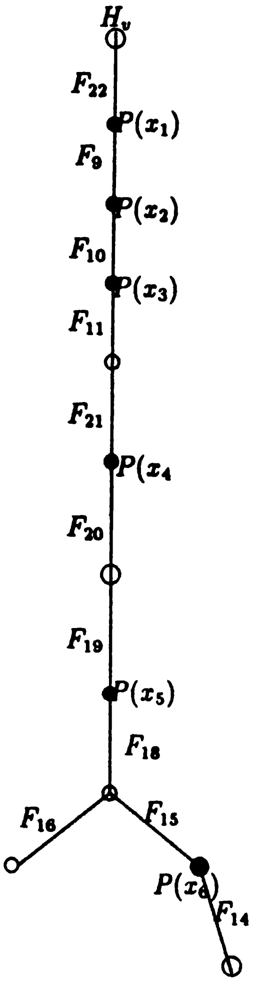

Fig. 2b

Finding the median for every tree maybe executed with the optimal (linear) complexity [2]. Information about the median of every tree determines the median of a polygon. In order to obtain the algorithm with optimal complexity for computing a median it is sufficient to oftain the trees H_0 and H_1 with the optimal complexity.

It is possible to do this job without complete description of a grid obtained from the initial polygon.

It is obvious that the number of edges of constructed trees is $O(n+k)$. For computing the weights of vertices of these trees the optimal method for point locating may be used ([5],[6]).

The median of any tree determines the set of strips. The median of the whole space is determined by the intersection of the union of horisontal strips with the union of vertical strips.

REFERENCES

1. V.Cepoi and F.Dragan, *Computing the median point of a simple rectilinear polygon.* (to appear)
2. P.Soltan, D.Zambitchii, Ch.Prisacaru, *Ekstremalinie zadaci na grafax*, Chisinau, Stiinta, 130 p., 1973.
3. V.Soltan, *"Vvedenie v aksiomaticeskuiu teoriu vîpuklosti"*, Chişinău, Ştiinţa, 223 pagini, 1984.
4. P.Soltan, Ch.Prisăcaru, *O razbienii ploscoi oblasti na d-vipuclosti* DAN SSSR, N2, T.262 1982.
5. H.Edelbrunner, L.J.Guibas and J.Stolfi. *Optimal point location in a monotone subdivizion.* SIAM J.

Comput., 15:317-340, 1985.

6. D.G.Kirkpatrick, *Optimal search in planar subdivizions*, SIAM J. Comput., 12:28-35, 1983.

STUDIA UNIV. BABEŞ-BOLYAI, MATHEMATICA, XL, 3, 1995

OPTIMALITY CONDITIONS FOR MULTIOBJECTIVE SYMMETRIC CONVEX PROGRAMMING

Ştefan ŢIGAN*

Received: 31.10.1995
AMS subject classification: 90C15, 90C39

REZUMAT. - Condiţii optimale pentru programarea neliniară multiobiectiv cu funcţii obiectiv pseudo-monotone şi simetric diferenţiabile. În această lucrare se prezintă condiţii de optim de tip Weber pentru o clasă de probleme de programare neliniară multiobiectiv cu funcţii pseudo-monotone şi simetric diferenţiabile, precum şi o condiţie de optim şi o teoremă de dualitate slabă pentru o problemă de max-min cu funcţii obiectiv simetric pseudo-convexe şi restricţii simetric quasiconvexe.

1. Introduction. In this paper for a class of nonlinear multiobjective programming problems with symmetrically differentiable pseudo-monotonic objective functions we present optimality conditions of Weber type [24].

We establish also a sufficient optimality condition and a weak duality theorem for a max-min problem involving symmetric pseudo-convex objective functions and symmetric quasi-convex constraints. In this aim, we transpose some of the results of Weir and Mond [25] to this symmetric pseudo-convex max-min problem.

2. Symmetric (generalized) convex functions. In this section we will briefly summarize some basic definitions and properties of symmetrically differentiable functions. Beyond this, some results concerning the so called symmetric pseudo and quasi-concave (convex) functions are considered. These classes of functions are generally nonlinear nonconcave and nondifferentiable. For further details we refer Minch [12]. Various properties of the usual pseudo and quasi-concave (or pseudo and quasi-convex) differentiable functions have been presented by Mangasarian [10], Martos [11], among others. Interesting results was

* *University of Medicine and Pharmacy, 3400 Cluj-Napoca, Romania*

obtained in the pseudo-monotonic case, from which we refer a Dantzig-Wolfe decomposition method for quasimonotonic programming [15], linearization procedures for pseudomonotonic programming [1], [2], [13], [16], optimality and duality properties [9], [19], [20], [22]. Some applications of these classes of functions in the max-min programming are given in [17], [18].

Another extensions of the quasi-convex and pseudo-convex functions are given by R. Pini and S. Schaible [25], and S. Komlosi [7], by using the generalized monotonicity. Also, G. Giorgi , A. Guerraggio [5], G. Giorgi and E. Molho [7] and G. Giorgi and S. Mititelu [6], present several observations on generalized invex functions and their relationships with other classes of generalized convex functions including the quasi-convex and pseudo-convex functions.

In [23], we considered symmetric invex functions and we extended some of the Giorgi and Molho [7] results for this more general class of generalized convex functions.

First we recall that for a real function f of one real variable the symmetric derivative of f at x is defined as:

$$f^s(x) = \lim_{h \to 0} (f(x+h) - f(x-h))/ 2h,$$

provided this limit exists (see, e.g. [12]).

This idea was extended by Minch [12] to functions of several variables.

DEFINITION 2.1 (Minch [12]) Let x be an element in an open domain A in R^n and let f:A --> R. If there exists a linear operator $f^s(x)$ from R^n to R, called the symmetric derivative of f at x, such that for sufficiently small h in R^n

$$f(x+h)-f(x-h) = 2\, f^s(x)\, h + u(x,h)\, \|h\| ,$$

where u(x,h) is in R and u(x,h) --> 0 as $\|h\|$ --> 0, then f is said to be symmetrically differentiable at x. If f has a symmetric derivative at each point x in A , then f is symmetrically differentiable on A.

The notions of symmetric gradient and symmetric derivative are analogous those of ordinary gradient and directional derivative. For convenience we shall denote the symmetric gradient of a symmetrically differentiable function f at x by $f^s(x)$.

Minch [12] shown that f is symmetrically differentiable at x in A , then symmetric gradient is of the form:

$$f^s(x) = (D^s f(x;e^1),\dots,D^s f(x;e^n)),$$

where $e^1,\dots,e^n$ is the natural basis for R^n and $D^s f(x;h)$ denote the symmetric derivative of f at x (in A) in the direction h (in R^n), that is :

$$(2.1)\quad D^s f(x;h) = \lim_{t \to 0} \frac{f(x+th) - f(x-th)}{2t} .$$

Let $f: A \to R$ and $g: A \to R$ be symmetrically differentiable functions at $x \in A$. From Definition 2.1, it follows easily, that:

i) f+g is symmetrically differentiable at x and

$$(2.2)\quad (f+g)^s(x) = f^s(x) + g^s(x);$$

ii) if f and g are continuous at x and g(x) is not equal with zero, then f/g is symmetrically differentiable at x and

$$(2.3)\quad (f/g)^s(x) = \frac{f^s(x)\, g(x) - f(x)\, g^s(x)}{g^2(x)}.$$

The following definition generalizes the pseudo-convexity concept.

DEFINITION 2.2 (Minch [12]) Let B be a subset of A and x' a point in A. The function f is said to be symmetrically pseudo-convex or s-pseudo-convex at x' (with respect to B) if f is symmetrically differentiable at x' and for all x in B,

$$f^s(x')\,(x-x') \geq 0 \text{ implies } f(x) \geq f(x').$$

The function f is s-pseudo-convex on A if it is s-pseudo-convex at each point of A.

The function f is s-pseudo-concave if -f is s-pseudo-convex .

Analogous to the ordinary notion of differentiable quasi-convexity it can be considered the notion of symmetrically quasi-convex function.

DEFINITION 2.3 (Minch [12]) Let B be a subset of A and x′ a point in A. The function f is said to be symmetrically quasi-convex or s-quasi-convex at x′ (with respect to B) if f is symmetrically differentiable at x′ and for all x in B ,

$f(x) \leq f(x')$ implies that $f^s(x')\,(x-x') \leq 0$.

The function f is s-quasi-convex on A if it is s-quasi-convex at each point of A. Also the function f is s-quasi-concave if -f is s-quasi-convex.

Examples:

1. The function f:R--->R defined by

$f(x) = x$, for $x < 1$,

$f(x) = 1$, for $x \in [1,2]$,

$f(x) = x-1$, for $x > 2$,

is a s-quasi-convex function but it is not s-pseudo-convex.

2. The function f_1:R--->R defined by

$f_1(x) = x$, for $x < 1$,

$f_1(x) = 0.5\,(x+1)$, for $x \in [1,3]$,

$f_1(x) = x-1$, for $x > 3$,

is both s-pseudo-convex and s-quasi-convex but it is not pseudo-convex.

3. The function f_2:R--->R defined by

$f_2(x) = x$, for $x < 1$,

$f_2(x) = 0$, for $x = 1$,

$f_2(x) = 0.5\,(x+1)$, for $x \in (1,3]$,

$f_2(x) = x-1$, for $x > 3$,

is s-pseudo-convex but it is not s-quasiconvex.

Next, it will be assumed that s-pseudo-convexity (or s-quasi-convexity) at a point is with respect to the definition domain of the function unless otherwise stated.

DEFINITION 2.4 (Minch [12]) Let B be a subset of A and let x' be a point in A. The function f is said to be s-pseudo-monotonic (s-quasi-monotonic) at x' (with respect to B) if is symmetrically differentiable at x' and both s-pseudo-convex and s-pseudo-concave (s-quasi-convex and s-quasi-concave).

Since , if f has an ordinary derivative at x , then f has a symmetric derivative at x and they are equal, the following property holds.

PROPOSITION 2.1 (i) If f is pseudo-convex (pseudo-concave) then f is s-pseudo-convex (s-pseudo-concave).

(ii) If f is differentiable quasi-convex (quasi-concave) then f is s-quasi-convex (s-quasi-concave).

(iii) If f is pseudo-monotonic (differentiable quasi-monotonic) then f is s-pseudo-monotonic (s-quasi-monotonic).

It is easy to see that the converse assertions of those stated in Proposition 2.1 are not true.

Next we give some useful properties of the symmetrically quasi and pseudo-convex functions.

PROPOSITION 2.2 (Tigan [22]) Let f be a symmetrically differentiable and continuous function. If f is a s-quasi-convex function on a convex subset B of A, then f is quasi-convex on B.

PROPOSITION 2.3 If f is s-pseudo-convex and continuous on a convex subset B of

A, then f is quasi-convex on B.

PROOF. Let x', x'' be two points in B such that $f(x') \leq f(x'')$. Suppose there exists x^* in the interval (x', x'') such that $f(x^*) > f(x'')$. Then, since f is continuous, there exists

$$x^0 = t'x' + (1-t')x'' \ , \ 0 < t' < 1,$$

such that

$$f(x^0) = \max \{ f(x) \mid x \in [x', x''] \}.$$

Therefore, by s-pseudo-convexity of f, because $f(x') < f(x^0)$ it follows that

$$(x' - x^0)\, f^s(x^0) < 0,$$

so, we have

(2.4) $(1-t')(x'-x'')\, f^s(x^0) < 0.$

Also, the inequality $f(x'') < f(x^0)$ implies that

(2.5) $(x''-x^0)\, f^s(x^0) = -\, t'(x'-x'')\, f^s(x^0) < 0.$

But (2.4) contradicts (2.5). Therefore f is quasi-convex on B.n

CONJECTURE 2.3.1 If f is s-pseudo-convex and continuous on a convex subset B of A, then f is s-quasi-convex on B.

3. Multiobjective symmetric pseudo-monotonic programming. Let f_k ($k \in I = \{1,2,\ldots,p\}$) be arbitrary objective functions defined on the open subset D of R^n and let X be a nonempty subset of D. Then we consider the following multiobjective programming problem:

VP. Find

(3.1) $\text{Vmax}\, (f_1(x),\ldots,f_p(x))$

subject to $x \in X$.

If f_k ($k \in I$) are s-pseudomonotonic objective functions then VP is said to be a symmetric pseudomonotonic multiobjective program. In (9.1), "Vmax" means that efficient points are regarded as optimal solutions to VP.

DEFINITION 3.1 A point $x^* \in X$ is said to be efficient solution for VP if and only if there does not exist another point $x' \in X$ such that :

$f_k(x') \geq f_k(x^*)$, for all $k \in I$ and

$f_{k'}(x') > f_{k'}(x^*)$ for at least one $k' \in I$.

The set of all efficient solutions to VP is denoted by E(X).

DEFINITION 3.2 A point $x^* \in X$ is said to be weakly efficient solution for VP if and only if there does not exist another point $x' \in X$ such that :

$f_k(x') > f_k(x^*)$, for all $k \in I$.

Clearly, every efficient point for a multiobjective program VP is weakly efficient but not conversely

As it is done e.g. by Bitran and Magnanti [3] (see, also [24]) we will relate the problem VP under the assumption of symmetric differentiability to a linear approximation at a point $x^0 \in X$ of that problem, namely

$P(x^0)$. Find

$$\text{Vmax} \; (f_1^s(x^0)\, x, \ldots, f_p^s(x^0)),$$

subject to $x \in X$.

The following Theorem 3.1 gives a fully symmetric relation between VP and $P(x^0)$. A similar result has shown to be true by Weber [24], who, however, restricted to the differentiable pseudomonotonic case, and which generalized a result obtained by Tigan [21] for the linear fractional multiobjective programming.

THEOREM 3.1 Let f_k ($k \in I$) be s-pseudomonotonic and continuous functions. A point $x^* \in X$ is efficient for the symmetric pseudomonotonic program VP if and only if x^* is efficient for $P(x^*)$.

PROOF. First, let $x^* \in X$ be efficient for VP. Then, there is no $x' \in X$ such that:

$f_k(x') \geq f_k(x^*)$, for all $k \in I$ and

$f_{k'}(x') > f_{k'}(x^*)$ for at least one $k' \in I$.

Let suppose there is $x' \in X$, such that

(3.2) $f_k^s(x') \geq f_k^s(x^*)$, for all $k \in I$ and

(3.3) $f_{k'}^s(x') > f_{k'}^s(x^*)$ for at least one $k' \in I$.

But since f_k $(k \in I)$ is s-pseudoconvex and hence it is s-quasi-convex, it results from (3.2) and (3.3) that

$f_k(x') \geq f_k(x^*)$, for all $k \in I$ and

$f_{k'}(x') > f_{k'}(x^*)$ for at least one $k' \in I$.

But this contradicts the fact that x^* is an efficient solution for $P(x^*)$.

Conversely, let $x' \in X$ be efficient for $P(x^*)$. Then there is no x' in X such that

(3.4) $f_k^s(x') \geq f_k^s(x^*)$, for all $k \in I$ and

(3.5) $f_{k'}^s(x') > f_{k'}^s(x^*)$ for at least one $k' \in I$.

By s-pseudo-concavity of f_k $(k \in I)$, from (3.4) and (3.5), we conclude that there is no x' in X such that

$f_k(x') \geq f_k(x^*)$, for all $k \in I$ and

$f_{k'}(x') > f_{k'}(x^*)$ for at least one $k' \in I$,

i.e. x^* is efficient for VP.n

THEOREM 3.2 Let f_k $(k \in I)$ be s-pseudo-monotonic and continuous functions. A point $x^* \in X$ is weakly efficient for the symmetric pseudo-monotonic multiobjective program VP if and only if x^* is weakly efficient for $P(x^*)$.

PROOF. The proof of this theorem is similar to that of Theorem 3.1 .n

4. Optimality conditions for symmetric pseudoconvex minimax problems. In this section, we consider the following minimax problem:

MP. Find

$$\underset{x}{\text{Min}}\ \text{Max}\ \{f_1(x),\dots,f_r(x)\}$$

subject to

$$g(x) \leq 0,$$

where $f_i:R^n \to R$, $(i=1,2,\dots,r)$ and $g:R^n \to R^m$ are symmetric differentiable functions (see, e.g. [11]).

The principal purpose of this section is to establish a sufficient optimality condition for problem MP involving symmetric pseudo-convex objective functions and symmetric quasiconvex constraints. We also define a dual problem to MP and establish a weak duality theorem. In this aim, we transpose some of the results of Weir and Mond [25] to the symmetric pseudo-convex maximin problem MP.

If the general minimax problem MP has a finite optimal value, then it may be expressed as following equivalent problem:

EP. Find

min q

subject to

$$f(x) \leq q.e$$

$$g(x) \leq 0,$$

where

$$f(x) = (f_1(x),\dots,f_r(x))^t,\ g(x) = (g_1(x),\dots,g_m(x))^t,$$

$$e = (1,1,\dots,1) \in R^r \text{ and } q \in R.$$

The main result of this section is:

THEOREM 4.1: Let f_i $(i=1,2,\dots,r)$ be s-pseudoconvex and g s-quasiconvex. If exist $x^* \in R^n$, $q^* \in R$, $v^* \in R^r$, $u^* \in R^m$, such that :

(4.1) $v^* f^s(x^*) + u^* g^s(x^*) = 0,$

(4.2) $v^* (f(x^*) - q^* e) = 0,$

(4.3) $u^* g(x^*) = 0,$

(4.4) $v^* \geq 0, \ v^* e = 1, \ u^* \geq 0,$

where $f = (f_1,...,f_r)$ and $e = (1,1,...,1) \in R^r$, then x^* is an optimal solution for problem MP.

In this theorem f^s denotes the symmetric gradient of the function f.

This theorem generalizes a similar result obtained by Weir and Mond [25] in the case of pseudo-convex objective functions and quasiconvex constraints.

PROOF. Suppose that (x^*,q^*) is not optimal solution for EP. Then there exists an feasible solution (x,q) for EP with $q<q^*$. Thus

$f_i(x) \leq q < q^*$, $i=1,2,...,r$

and hence

$v^*_i f_i(x) \leq v^*_i q^*$, $i=1,2,...,r$

with at least one strict inequality, since by (4.4), v^* is not the nul vector. Hence, by (4.2),

$v^*_i f_i(x) \leq v^*_i f_i(x^*)$, $i=1,2,...,r$

with at least one strict inequality.

Since f_i is assumed s-pseudo-convex, then, for each $i=1,2,...,r$ and $v_i \geq 0$, $v_i f_i$ is s-pseudo-convex and

$(x-x^*)^t (v^*_i f^s_i(x^*)) \leq 0$, $i=1,2,...,r$

with at least one strict inequality.

Hence

$(x-x^*)^t (v^{*t} f^s(x^*)) < 0.$

Then it follows from (4.1) that

(4.5) $(x-x^*)^t (u^{*t} g^s(x^*)) > 0.$

From (4.3), since x is feasible for EP, it results

$$u^*_i\, g_i(x) - u^*_i\, g_i(x^*) \le 0,\ i=1,2,\ldots,m.$$

But symmetric quasiconvexity of g implies

$$(x-x^*)^t\, (u^*_i\, g_i^s(x^*)) \le 0,\ i=1,2,\ldots,m$$

and hence

$$(x-x^*)^t\, (u^{*t}\, g^s(x^*)) \le 0$$

which contradicts (4.5).

Thus (x^*,q^*) is optimal for EP and x^* is optimal for MP.n

In relation to MP, which is equivalent to EP, we consider the following dual program:

DMP. Find

$$\max z$$

subject to

(4.6) $v_i\, (f_i(y)-z) \ge 0,\ i=1,2,\ldots,r$

(4.7) $v^t\, f^s(y) + u^t\, g^s(y) = 0$

(4.8) $u^t\, g(y) \ge 0$

(4.10) $v \ge 0,\ v^t e = 1,\ u \ge 0,\ z \in R.$

THEOREM 4.2 (Weak Duality) Let (q,x) be a feasible solution for EP and let (y,v,u,z) be a feasible solution for DMP. If f is s-pseudo-convex and, for all feasible (q,x,y,v,u,z) the function $u^t g$ is s-quasiconvex then $q \ge z$.

PROOF. Suppose $q < z$. Then

$$f_i(x) < v,\ i=1,2,\ldots,r$$

and, therefore

$$v_i\, (\, f_i(x) - z\,) \le 0,\ i=1,2,\ldots,r$$

with at least one strict inequality, since by (4.10), v is not the nul vector. From (4.6)

$$v_i\, f_i(x) \le v_i\, f_i(y),\ i=1,2,\ldots,r$$

with at least one strict inequality.

Since each f_i is s-pseudo-convex, it follows

$$(x-y)^t (v_i f^s_i(y)) \leq 0, \; i=1,2,...,r$$

with at least one strict inequality.

Therefore

$$(x-y)^t (v^t f^s(y)) < 0$$

and from (4.7)

(4.11) $(x-y)^t (u^t g^s(y)) > 0.$

From feasibility of x for EP and from (4.8) and (4.9)

$$u^t g(x) - u^t g(y) \leq 0$$

and since $u^t g$ is s-quasi-convex

$$(x-y)^t (u^t g^s(y)) \leq 0$$

which contradicts (4.11).n

5. Conclusions. For a class of multiobjective programming problems with symmetrically differentiable objective functions we present optimality conditions of Weber type.

We generalize also some results of Weir and Mond [25], establishing a sufficient optimality condition and a weak duality theorem for a max-min problem involving symmetric pseudo-convex functions and symmetric quasi-convex constraints.

Finally, we note that some of Weber's results [24] concerning the linearization techniques for finding efficient solutions of pseudo-monotonic multiobjective programming with linear constraints can be extended to the symmetrically pseudo-monotonic case.

REFERENCES

[1] Bector C.R., Jolly P.L., Programming problems with pseudomonotonic ojectives, Optimization, 15 (1984),

2, 217-219.

[2] Bhatt S.L., Linearization Technique for linear fractional and pseudomonotonic programs revisited, Cahiers du CERO, 23 (1981), 53-56.

[3] Bitran G.R., Magnanti T.L. , The structure of admissible points with respect to cone dominance, JOTA, 29 (1979), 573-614.

[4] Dantzig G.B., Linear programming and extensions, Princeton University Press, Princeton, New-Jersey, 1963.

[5] Giorgi G., Guerraggio A., Various types of invex functions, Dipartimento di Richerche Aziendali, Universita di Pavia, 1994.

[6] Giorgi G., Mititelu S., Invexity in nonsmooth programming, Atii del Tredicesimo Convegno A.M.A.S.E.S., Verona, 1989, 509-520.

[7] Giorgi G., Molho E., Generalized invexity: Relationships with generalized convexity and applications to optimality and duality conditions, in Proceedings of the Workshop held in Pisa, 1992, "Generalized Concavity for Economic Applications", ed. Piera Mazzoleni, 1992, 53-70.

[8] Komlosi S., Generalized Monotonicity of generalized Derivatives, in Proceedings of the Workshop held in Pisa, 1992, "Generalized Concavity for Economic Applications", ed. Piera Mazzoleni, 1992, 1-6.

[9] Kortanek K.O., Evans J.P., Pseudo-concave Programming and Lagrange regularity, Oper. Res., 15 (1967), 882-891.

[10] Mangasarian O.L., Nonlinear Programming, New York et al., Mc Graw Hill,1969.

[11] Martos B., Nonlinear Programming Theory and Methods, Amsterdam-Oxford, North-Holland,1975.

[12] Minch R. A., Applications of symmetric derivatives in mathematical programming, Math. Prog., 1 (1971), 307-320.

[13] Mond B., Techniques for pseudo-monotonic programming, LaTrobe University, Pure Math. Res. Paper No.82-12, Melbourne, 1982.

[14] Pini R., Schaible S.,Some Invariance Properties of Generalized Monotone Maps,in Proceedings of the Workshop held in Pisa, 1992, "Generalized Concavity for Economic Applications", ed. Piera Mazzoleni, 1992, 87-88.

[15] Tigan S., Sur une methode de decomposition pour le probleme de programmation monotone, Rev. Analyse Numer. Theor. Approx., 12, 1 (1983), 347-354.

[16] Tigan S., On the linearization technique for quasi-monotonic optimization problems, Analyse Num. Theor. Approx., 12,1 (1983), 89-96.

[17] Tigan S., A quasimonotonic max-min programming problem with linked constraints, Itinerant Seminar on Functional Equations, Approximation and Convexity, Cluj-Napoca University, 1986, 279-284.

[18] Tigan S., On a quasimonotonic max-min problem, Analyse Numer. Theor. Approx., no.1 (1990), 85-91.

[19] Tigan S., On duality for generalized pseudomonotonic programming, Analyse Num. Theor. Approx., 20, 1-2 (1991), 111-116.

[20] Tigan S., On Kortanek-Evans optimality conditions for symmetric pseudo-concave programming, Itinerant Seminar on Functional Equations, Approximation and Convexity, Cluj-Napoca University, 1992.

[21] Tigan S., Sur le probleme de la programmation vectorielle fractionnaire, Analyse Numer. Theor. Approx., 4, 1 (1975), 99-103.

[22] Tigan S., Linearization procedure and Kortanek-Evans optimality conditions for symmetric pseudo-concave programming, Analyse Num. Theor. Approx., 22, 1 (1993), 113-120.

[23] Tigan S., Optimality conditions for symmetric generalized convex programming and applications, Studii si

Cerc. Mat., 46,4 (1994).

[24] Weber R., Pseudomonotonic Multiobjective Programming, Discussion Papers B8203, Institute of Operations Research, Univ. of Saarland, Saarbruecken, 1982.

[25] Weir T., Mond B., Sufficient optimality conditions and duality for a pseudoconvex minimax problem, Cahiers du CERO, 33, n.1-2 (1991), 123-128.

STUDIA UNIV. BABEŞ-BOLYAI, MATHEMATICA, XL, 3, 1995

SOME ASPECTS OF GRAPHS PLANARITY

Teodor TOADERE, Florin STOICA*

Received: 15.12.1994
AMS number classfication: 05C45, 05C85, 68Q20, 68R10

REZUMAT: - Câteva aspecte ale planarităţii grafelor. Prezentăm un algoritm de testare a planarităţii grafelor hamiltonienc. Slgoritmul este dat şi pentru testarea grafelor 4-convexe şi 3-convexe.

1. Introduction. The aim of this paper is to present an algorithm (and its Pascal language version) for testing the planarity of hamiltonian graphs. For a certain planar hamiltonian graph we will build a planar representation of it.

In the first part some theoretical results will be presented. These results will lead us to the fundamental idea applied in the algorithm. As some known results show us, the algorithm may be used in order to test the planarity of 4-connected graphs and 3-connected graphs with at most an articulation set having the cardinal 3.

Among different theoretical results, the notion of bridge of a subgraph is widely used. Thus, the concept of overlapped bridges plays an important role for the theory of plane representation of graphs [3].

The bridges were very used for the investigation of planar graph cycles. Important results have been obtained by Tutte, Thomasson, Nelson.

The concept of bridge was also successfully used for studying the properties of graphs with respect to connectedity.

In the scientific literature the "bridge method" is used as a method to prove different graph theory theorems.

2. Basic concepts. In this paper we will only talk about finite, undirected graphs, with no loops and with no multiple edges.

* *"Babeş-Bolyai" University, Faculty of Mathematics and Computer Science, 3400 Cluj-Napoca, Romania*

Let us denote MxM by $M^{(2)}$, and let G=(V,E) such a graph. The graph G'=(V',E') is a subgraph of the graph G=(V,E) if $V'\subseteq V$ and $E'\subseteq E$. Thus, we will write $G'\subseteq G$. We remark that $M\subset N$ means $M\subseteq N$ and $M\neq N$, M and N being finite sets.

If $W\subseteq V$, then the graph $(W,E\cap W^{(2)})$ is a subgraph of G=(V,E), namely the subgraph induced by W. We denote this subgraph by G(W).

If $E'\subseteq E(G)$, then G-E' denotes the graph produced from G by eliminating the edges from E'. So, G-E'=(V(G),E(G)-E').

If $W\subseteq V(G)$, then G-W is the graph produced from G by eliminating the vertices from W (obviously, if x is eliminated from W, so will be the edges incident to x). So, $G-W=(V-W,E\cap(V-W)^{(2)})$.

If W={x} we write G-x instead of G-{x}. Similary, if $E'=\{e\}\subseteq E$ we write G-e instead of G-{e}.

If $H\subseteq G$ (H is a subgraph of G) we may write G-H instead of G-V(H).

If $e\in V^{(2)}-E$, then the graph produced from G by adding the edge e is $G\cup\{e\}$ or $G\cup e$.

Let x and y be two vertices of G, not necessarily distinct. By x,y-chain we mean an alternant sequence of vertices and edges $x_1,e_1,x_2,e_2,\ldots,x_k,e_k,x_{k+1}$, where $x=x_1$, $y=x_{k+1}$ and $e_i=(x_i, x_{i+1})\in E(G)$, $1\le i\le k$. The x,y-chain may also be denoted by $W=x_1x_2\ldots x_{k+1}$, or $W=[x_1,x_2,\ldots,x_{k+1}]$.

The chain above is an elementary x,y-chain if all its vertices are distinct. The elementary x,x-chain has the length equal to zero and is made up only by the vertex x.

In what follows, when we will say cycle, we will mean an elementary cycle.

If $\mu=x_1\ldots x_k$ is an elementary chain, $P=x_i$, $Q=x_j$ and $1\le i\le j\le k$, then the P,Q-segment of μ is the elementary P,Q-chain $x_i\ldots x_j$, and we will approach it as a partial chain of μ, with the limits P and Q. This segment will be denoted by $\mu[P,Q]$, and $\mu[P,Q]-\{P,Q\}$ will be denoted by $\mu(P,Q)$.

3. The bridges of a subgraph H. Let H be a subgraph of the graph G=(V(G),E(G)).

Definition 3.1 An H-bridge in G is a subgraph of G-E(H) which is either an edge (with its limits), linking the two vertices of H, or a connected component K of G-H, with all the edges (and their limits) of G incident to K.

The H-bridges from the former category will be named singular or diagonal bridges of H, and the others (non-singular) will be named regular bridges.

In the figure 3.1, B_i, $i\in\{1,2,3,4\}$, are regular H-bridges, and B_5 is a singular H-bridge, where H is the cycle represented by the dotted line.

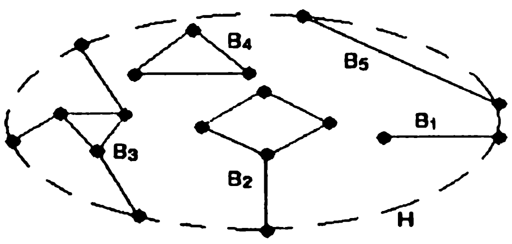

Figura 3.1

Definition 3.2 If B is an H-bridge, the vertices from $V(B)\cap V(H)$ are named supporting vertices of B, and the vertices of $V(B)\setminus V(H)$ are named inner vertices of B.

We notice that the H-bridges B_i, $i=1,\ldots,5$ from figure 3.1 have 1,1,3,0,2 supporting vertices respectively, and 1,4,3,3,0 inner vertices respectively.

Definition 3.3 The kernel of an H-bridge is the subgraph induced by the inner vertices of B.

We remark that the kernel of a singular bridge is the empty graph.

Lemma 3.1 If B_1 and B_2 are two H-bridges, $B_1\neq B_2$, then the kernels of the two bridges do not have common vertices, i.e.

$$(V(B_1)\setminus V(B_2))\cap(V(B_2)\setminus V(B_1))=\varnothing.$$

Proof: If B_1 or B_2 are singular, the lemma holds because in the former case $V(B_1)\setminus V(B_2)=\varnothing$ and in the latter $V(B_1)\setminus V(B_2)=\varnothing$.

Let us suppose that B_i is a regular H-bridge, $i=1,2$, and that the kernels of the two bridges have at least a common vertex. From the definition 3.1, the kernel of B_i is a connected component K_i of G-H, $i=1,2$. Then it would result that $K_1=K_2$, and, moreover, $B_1=B_2$. This contradicts the hypothesis and concludes the proof.

Remark 3.1 Two H-bridges may have in common only supporting vertices.

. **4. Overlap graphs and circle graphs**. In what follows we will consider only the bridges of the cycles.

Let B be a bridge of a cycle C in the graph G, having the supporting vertices $a_1,\dots,a_s$, $s\geq 2$, which are on C in this cyclic order. The s segments of C, denoted by $C_B[a_i,a_{i+1}]$, $1\leq i\leq s-1$ and $C_B[a_s,a_1]$, are named segments of C with respect to the C-bridge B.

Definition 4.1 Let B and B' be two C-bridges in the graph G. B and B' are parallel if and only if there exist two vertices x and y on C so that all the supporting vertices of B are included in the segment C[x,y] and all the supporting vertices of B' are included in the segment C[y,x].

In figure 4.1, the C-bridges B_1 and B_3 are parallel, where C is the cycle represented by the dotted line.

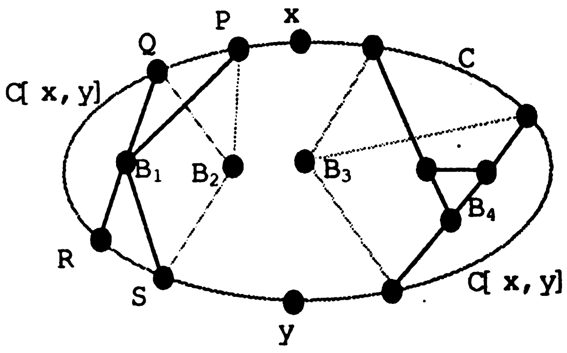

Figura 4.1

Remark 4.1 If B or B' has at most a supporting vertex, then B and B' are parallel.

Remark 4.2 Let us suppose that the C-bridge B has at least two supporting vertices. These vertices subdivide C into segments with respect to B. Then, B and B' are parallel if and only if either B' does not have supporting vertices or there exists a segment of C with respect to B which includes all the supporting vertices of B'.

Definition 4.2 Two C-bridges B and B' overlap if and only if B and B' are not parallel.

The pairs (B_1, B_2) and (B_3,B_4) from figure 4.1 are examples of overlapping C-bridges.

Definition 4.3 Two C-bridges B and B' are crossed if and only if there exist four vertices P, Q, R, S in this cyclic order on the cycle C, so that P and R are in V(B) and Q and S are in V(R').

An example of crossed bridges is the pair (B_1, B_2) of C-bridges from figure 4.1.

Lemma 4.1 Let C be a cycle of the graph G and B and B' two C-bridges. The following statements are equivalent:

(1) B and B' overlap;

(2) B and B' are crossed or they have exactly three supporting vertices each, and those are identical.

Proof: (1) ==> (2)

Let $a_1, \dots, a_s$, $s>=2$ the supporting vertices of B_1, on this cyclic order on C (because B and B' overlap, each of them has at least two supporting vertices). We denote $a_{s+1}=a_1$. Let $C_B[a_i,a_{i+1}]$, $1<=i<=s$ the segments of C with respect to the C-bridge B.

Case (a): Let us suppose that there exists a number k in $\{1,\dots,s\}$ so that $V(B')\cap(C_B(a_k,a_{k+1})\neq\varnothing$.

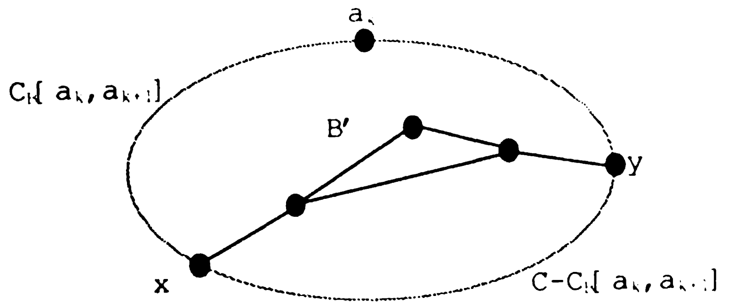

Figura 4.2

Let $x\in V(B')\cap(C_B(a_k,a_{k+1})$. Because B and B' overlap, there exists a supporting vertex y of B' in $C-C_B[a_k,a_{k+1}]$ (otherwise, all the supporting vertices of B' would be in $C_B[a_k, a_{k+1}]$, and thus B and B' would be parallel). Then, a_k, x, a_{k+1}, y are on C in this cyclic order (see fig. 4.2) and a_k, $a_{k+1}\in V(B)$ and $x,y\in V(B')$. From definition 4.3 it results that B and B' are crossed.

Case (b): Let us suppose that there is no k in $\{1,\dots,s\}$ so that $V(B')\cap(C_B(a_k,a_{k+1})\neq\varnothing$, i.e. for every k in $\{1,\dots,s\}$, $V(B')\cap(C_B(a_k,a_{k+1})=\varnothing$, Let us denote by S_B the set of supporting vertices of B and by $S_{B'}$ the set of supporting vertices of B'. From our supposition we have $S_{B'}\subseteq S_B$.

If $S_{B'} \subset S_B$ then there exists an l in {1,...,s} so that $a_l \notin S_{B'}$. With no lack of generality we may take l = 1. Then let i be the lowest index for which $a_i \in S_{B'}$, and let j be the greatest index for which $a_j \in S_{B'}$, with i and j in {2,...,s}. Thus, there exists a segment of C with respect to B', namely $C_{B'}[a_j,a_i]$, that fulfills $V(B) \cap (C_{B'}(a_k,a_{k+1})) \neq \emptyset$, and we are in the case (a). So, B and B' are crossed.

Let us now suppose that $S_B=S_{B'}$. Let us denote $p = |S_B|$. If p=2 then B and B' are parallel. It results p >= 3. If p=3 we may apply the lemma. If p >= 4, then B and B' are crossed.

(2) ==> (1) Let us firstly consider the case when B and B' have exactly three supporting vertices each, and these are identical.

It is obvious that there exists no segment of C with respect to B that should contain all the supporting vertices of B'. From the remark 4.2 we deduce that B and B' are not parallel, so B and B' overlap.

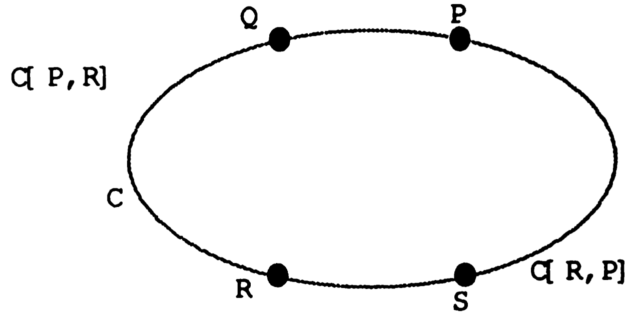

Figura 4.3

In the other case B and B' are crossed, i.e. there exist four vertices P,Q,R,S in that cyclic order on C so that P and R are in V(B) and Q and S are in V(R') (fig. 4.3). Let us suppose that B and B' are parallel. So, there exists a segment of C with respect to B that contains all the supporting vertices of B'. Let $C_B[x,y]$ be this segment.

So, $Q,S \in C_B[x,y]$. It results that $C[Q,S] \subseteq C_B[x,y]$ or $C[S,Q] \subseteq C_B[x,y]$. But, C[Q,S] contains the vertex R of B and C[S,Q] contains the vertex P of B. So, between the supporting vertices of B there exists at least a supporting vertex of B, and this contradicts the definition of the segments of C with respect to the C-bridge B.

Definition 4.4 Let C be a cycle of the graph G. We consider the C-bridges as the vertices of a new graph O(G:C), that will be called overlap graph of G with respect to C.

There exists an edge between two vertices B and B' of O(G:C) if and only if B and B' overlap.

Definition 4.5 If C is a hamiltonian cycle, the graph O(G:C) is called overlap graph or circle graph of G with respect to G.

Remark 4.3 If C is a hamiltonian cycle all the C-bridges are diagonals of C.

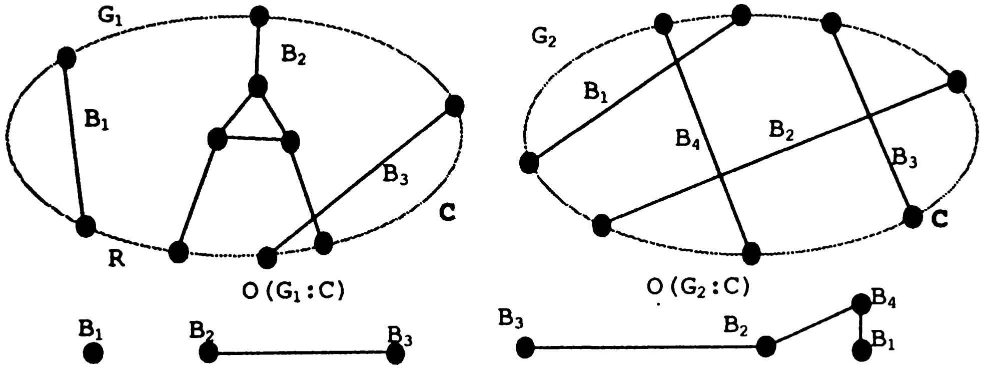

Figura 4.4

In fig. 4.4 are represented two graphs with their overlap graphs. The overlap graph O(G_2:C) is an example of circle graph.

Let us consider a hamiltonian graph G and a hamiltonian cycle C of it. Let G' be a geometrical representation of G in the plane such that C is represented by a simple closed Jordan curve C' and all the diagonals of C are represented by simple Jordan curves situated in the finite region of C'. The number of the intersection points of the Jordan curves that represent the diagonals of C equals the number of edges in the circle graph O(G:C). If C' is chosen as a geometrical circle, and its diagonals are represented by straight line segments, then we have a second definition of a circle graph:

Definition 4.6 The vertices of a circle graph are chords of the geometrical circle, and two chords are linked by an edge if and only if they intersect in a point interior to the circle.

4.1 Parallel bridges.

Lemma 4.1.1 Let C be a cycle of the graph G and B, B' two C-bridges. If B and B' are parallel then $|V(B) \cap V(B')| <= 2$ and the common vertices, if they exist, are supporting vertices.

Proof. From the remark 3.1, two C-bridges may have in common only supporting vertices. If $|V(B) \cap V(C)| <= 1$ or $|V(B') \cap V(C)| <= 1$, the statement in the lemma is true.

Now we suppose that both B and B' have at least two supporting vertices. From remark 4.2 it results that there exists a segment of C with respect to B that contains all the supporting vertices of B'. Let this segment be $C_B[x,y]$, where $x, y \in S_B$. Then, we have $V(B) \cap V(B') \subseteq \{x,y\}$ and so $|V(B) \cap V(B')| <= 2$ and that concludes the proof.

Proposition 4.1.1 Let C be a cycle of the graph G and B and B' two diagonals of C. Obviously, $|V(B) \cap V(B')| <= 1$. If $|V(B) \cap V(B')| = 1$, B and B' are parallel.

Proposition 4.1.2 Let C be a cycle of the graph G and B and B' two diagonals of C. If B and B' are crossed

then $|V(B) \cap V(B')| = 0$.

Because $|V(B)|=|V(B')|=2$, the statements above are (almost) obvious.

We state the following remark:

Remark 4.1.1 **Let us denote by $\mathcal{B}$ the set of C-bridges from a graph G. We define the binary relation r as:**

B r B' <==> B is parallel with B', for all B and B' in $\mathcal{B}$

The relation r is not an equivalence relation on $\mathcal{B}$. Even if r is reflexive and symmetrical, it is not transitive.

As we see in fig. 4.1.1., B_1 r B_2 and B_2 r B_3, but B_1 r B_3 fails (B_1 and B_3 are crossed C-bridges).

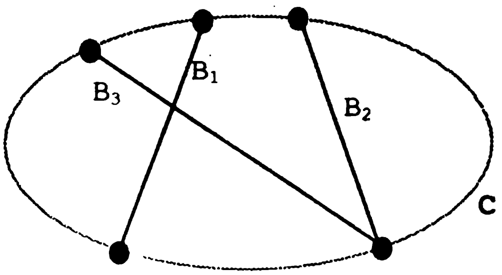

Figura 4.1.1.

4.2 Connected overlap graphs. We know that any C-bridge with at most a supporting vertex is parallel with any other bridge. This implies that any bridge with this property is an isolated vertex of the overlap graph O(G:C). Consequently, if O(G:C) is connected with |V(O(G:C))| >= 2, then every C-bridge has at least two supporting vertices.

Lemma 4.2.1 Let C be a cycle of the graph G, K a connected subgraph of O(G:C) and x and y two vertices of C. Let us suppose that K has a supporting vertex on each of the segments C(x,y) and C(y,x) (1). Then there exists a bridge B of K with a supporting vertex on each of the segments (2).

Remark A rigorous statement would be:

(1) C-bridges from G, considered as vertices of K, have supporting vertices on both of the segments C(x,y) and C(Y,x).

(2) There exists a C-bridge B, $B \subseteq G$, $B \in V(K)$, with at least a vertex on each of the segments above.

In what follows we will use for simplicity, the same style of statement.

Proof of the Lemma 4.2.1: Let a', a" be the supporting vertices of K on C(x,y) and C(y,x). Let us suppose that a' and a" are supporting vertices of the C-bridges B' and B", where B' and B" are from V(K). But, K is connected, so there exists an elementary B',B"-chain [$B'=B_1,\ldots,B_q=B''$] in O(G:C). Because B" has a supporting vertex on C(y,x), there exists an index i such that B_i has a supporting vertex in C(y, x). Let us denote by i the smallest index with the above property.

If i=1 then with B := B' the lemma is true.

If i>1 then B_{i-1} has no supporting vertex in C(y,x). Because B_{i-1} and B_i overlap (they are succesive vertices in the elementary B',B"-chain from O(G:C)), there results that B_i has a supporting vertex in C(y,x) (otherwise B_i and B_{i-1} are parallel). With B := B_i, the lemma is true.

For sets K of C-bridges (or subgraphs of O(G:C)), we define the segments of C with respect to K.

Let K be a set of C-bridges with at least two distinct supporting vertices, $a_1, \ldots, a_s$, s >= 2, on C in this cyclic order. Then the s segments denoted $C_K[a_i, a_{i+1}]$, 1<=i<=s-1 and $C_K[a_s,a_1]$ are called the segments of C with respect to K.

Let K and K' be two disjoint sets of C-bridges.

K and K' are parallel if and only if either one of them has at most a supporting vertex, or both of them have at least two supporting vertices, and a segment of C with respect to K contains the set of supporting vertices of K'.

K and K' overlap if and only if they are not parallel.

K and K' are crossed if and only if there exist four vertices P, Q, R, S in this cyclic order on C, so that P and R are in K and Q and S are in K'.

Lemma 4.2.2 Let C be a cycle of the graph G. If K and K' are disjoint sets of C-bridges, the following statements are equivalent:

(i) K and K' overlap;

(ii) K and K' are crossed or they have exactly three supporting vertices each, and these coincide.

The proof is analogous to that given for the lemma 4.1.

Theorem 4.2.1 Let C be a cycle of the graph G. Let K and K' be two connected subgraphs of O(G:C), not connected by any edge. Then, K and K' are parallel.

Proof. We prove the theorem by reduction to absurd. We suppose that K and K' overlap. From the lemma 4.2.2 we have two cases:

(i) K and K' are crossed;

(ii) K and K' have exactly three supproting vertices each, and those coincide.

Case (i): K and K' are crossed. Then from definition, there exist four vertices P, Q, R, S in this cyclic order on C, so that P and R are in K and Q and S are in K'. From lemma 4.2.1 there exists a bridge B in K having a vertex P' on C(S,Q) and a vertex R' on C(Q,S) (see fig. 4.2.1).

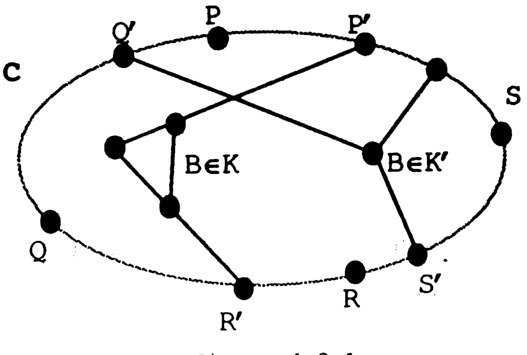

Figura 4.2.1.

From here it results that K' has the vertex Q on C(P',R'), and the vertex S on C(R',P'). From lemma 4.2.1 there exists a bridge B' in K' having a vertex Q' on C(P',R') and a vertex S' on C(R',P'). So, P',Q',R',S' are four vertices in this cyclic order on C, with P' and R' in B and Q' and S' in B'. It follows that the C-bridges B and B' are crossed, and so they overlap. Then, (B,B') is an edge that connects K with K' in O(G:C), and this contradicts the hypothesis.

Case (ii): K and K' have the same supporting set with three distinct vertices, x, y and z.

If K is an isolated vertex B of O(G:C), then x, y and z will be the supporting vertices of B.

If K has q >= 2 C-bridges $B_1, \ldots, B_q$, because K is connected every bridge B in K has the property that there exists a B^* in K, $B^* \neq B$, so that B and B^* overlap. But they cannot be crossed, so they have the same set of three supporting vertices, namely {x, y, z}.

Using the same logic for K', we have proved that all the bridges from K∪K'have the same set of three supporting vertices, {x, y, z}. But then the subgraph of O(G:C) induced by the bridges from K∪K' is complete. There results that K and K' are connected by an edge of O(G:C), and that contradicts the hypothesis.

In order to give a characterization of connected overlap graphs, we need the following definition:

Definition 4.2.1 A 2-separation (G_1, G_2) of the graph G contains two subgraphs G_1 and G_2 of G having at least three vertices and fulfilling the following conditions:

(i) $G = G_1 \cup G_2$;

(ii) $| V(G_1) \cap V(G_2) | = 2$;

(iii) $| E(G_1) \cap E(G_2) | = 0$.

Theorem 4.2.2 Let C be a cycle of the graph G. Then the overlap graph O(G:C) is not connected if and only if there exists a 2-separation (G_1, G_2) of G so that:

(i) $\{x,y\} := V(G_1) \cap V(G_2) \subseteq V(C)$;

(ii) $C[x,y] \subseteq G_1$, $C[y,x] \subseteq G_2$;

(iii) neither G_1, nor G_2 is a segment of C.

Proof: (<==) We suppose that there exists a 2-separation (G_1, G_2) of G satisfying the properites (i), (ii), (iii). Because the subgraph G_i is not a segment of C, with i=1,2, there results that G_i contains a C-bridge B_i, i=1,2.

Let us suppose that

$$S_{B1} \cap V(C(x,y)) = \varnothing. \qquad (*)$$

Then $S_{B1} \subseteq \{x,y\}$. Let us prove this. If B_1 is singular, this is obvious. Otherwise, B_1 is a regular C-bridge. Let us suppose that there exists u in S_{B1} such that

$$u \notin \{x,y\} \qquad (**)$$

From (*) and (**) we have that $u \in V(C(x,y))$. But, from (22) we have $C[x,y] \subseteq G_2$. So, $u \in V(G_1)$. Let v be an interior vertex of B_1. Obviously, $v \in V(G_1)$. From (1) we have that $v \notin V(G_2)$. We have obtained so far that

$$v \in V(G_1) \text{ and } v \notin V(G_2)$$
$$u \notin V(G_1) \text{ and } u \in V(G_2)$$

Since B_1 is connected there exists an elementary v,u-chain in B_1. From the definition of the 2-separation, $|E(G_1) \cap E(G_2)| = 0$. So, the edges of the elementary v,u-chain are either edges of G_1, or edges of G_2. In both of the cases there exists an edge in $E(G_i)$ (of the elementary chain) with the property that an extremity of it is not in $V(G_i)$, i=1,2, and that's impossible. So, $S_{B1} \subseteq \{x,y\}$. But then B_1 is parallel with any other C-bridge in G, so it is an isolated vertex of O(G:C). From the hypothesis (*), the statement to be proven is valid.

Let us now suppose that

$$S_{B2} \cap V(C(x,y)) = \varnothing.$$

By rationing in the same way we will conclude that O(G:C) is not connected.

Let us now suppose that

$$S_{B1} \cap V(C(x,y)) \neq \varnothing,$$

and

$$S_{B2} \cap V(C(x,y)) \neq \varnothing.$$

If B_1 and B_2 are members of the same component K of O(G:C), then from lemma 4.2.1 it results that the component K contains a C-bridge having at least a supporting vertex on each of the segments C(x,y) and C(x,y). Similarly we have that there exists an elementary chain with an extremity in $V(G_1)$ - $\{x,y\}$ and the other extremity in $V(G_2)$ - $\{x,y\}$, with the edges are either edges of G_1, or edges of G_2, and that is impossible.

Consequently, B_1 and B_2 are in different connected compenents of O(G:C), and so O(G:C) is not connected.

(==>)Let us suppose that O(G:C) is not connected. We consider two cases:

(a) G contains a C-bridge B with at most a supporting vertex. Then B is a component of O(G:C), namely an isolated vertex. (B is parallel with any other C-bridge).

Let us consider that B does not have supporting vertices. Let us denote

$$G_1=(V(B),E(B)\cup\{(x,y)\})$$

and

$$G_2=G-\{(x,y)\}-B,$$

where (x,y) is an edge of the cycle C. Then, (G_1,G_2) is a 2-separation of G that satisfies (1), (2), (3).

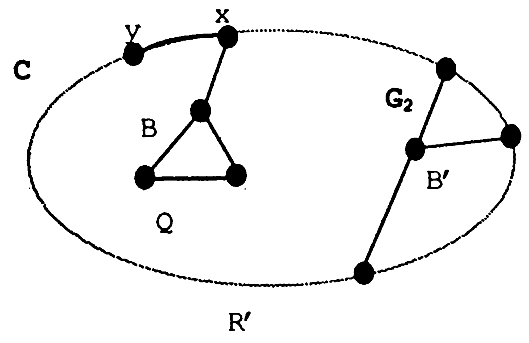

Figura 4.2.2.

Remark: The property (3), especially the fact that G_2 is not a segment of C, is less obvious. But O(G:C) is not connected, B being an isolated vertex in O(G:C). So, there exists a C-bridge B' in G, B'≠B. From definition of G_2 we have that B' is a subgraph of G_2, and so G_2 is not a segment of C (see figure 4.2.2).

If B has a single supporting vertex, namely a, we chose an edge (x,y) of C so that x=a.

Let us denote

$$G_1=(V(B)\cup\{x,y\},E(B)\cup\{(x,y)\})$$

and

$$G_2=G-\{(x,y)\}-(B-\{x\}).$$

Then, (G_1, G_2) is a 2-separation of G satisfying (1),(2),(3).

(b) G contains only C-bridges with at least two supporting vertices. Let us suppose that O(G:C) has p connected components, with p >= 2. Let these be $K_1, \ldots, K_p$. From Theorem 4.2.1 we deduce that K_1 and K_2 are parallel. There results that all the supporting vertices of K_2 are on a segment of C with respect of K_1. Let this segment be $C_{K1}[x,y]$.

Case (1). $(x,y) \in E(G)$. We denote with B the diagonal of C determined by (x,y).

(1.a) B is in K_1. Since K_1 is a connected component of O(G:C), we have that $K_1=\{B\}$, seen as a set of C-bridges, or K_1 is an isolated vertex (B) in O(G:C). So, we chose $G_1=C[x,y] \cup \{ K_i \mid K_i$ has all the supporting vertices on $C[x,y]$, $i=2,\ldots,p\}$ and $G_2=G(A)$, where $A=(V(G)\setminus V(G_1))\cup\{x,y\}$. Thus, we assured that in G_2 there exists at least a C-bridge B.

(1.b) B is not in K_1. We will show that B is an isolated vertex in O(G:C). We will supose that there exists a C-bridge B_1 so that B_1 and B are crossed. Let x_1, y_1 supporting vertices of B_1, with $x_1 \in C(x,y)$ and $y_1 \in C(y,x)$. But, then $x \in C(y_1,x_1)$ and $y \in C(x_1,y_1)$. From the lemma 4.2.1 there exists a bridge B_2 of K_1 having at least a supporting vertex on each of the segments $C(x_1,y_1)$, $C(y_1,x_1)$. So, B_1 and B_2 are crossed, and thus B_1 is in K_1, and x_1 is supporting vertex of K_1 on the segment $C_{k1}[x,y]$. This is a contradiction, and from here we have that B is parallel with any other C-bridge, and thus it is an isolated vertex in O(G:C). We chose $G_1=C[x,y] \cup \{ K_i \mid K_i$ has all the supporting vertices on $C[x,y]$, $i=1,\ldots,p\}$ and $G_2=G(A)\setminus(x,y)$, where $A=(V(G)\setminus V(G_1))\cup\{x,y\}$. Case (2). $(x,y) \notin E(G)$. We chose G_1 the same way as in the case (1.b) and $G_2=G(A)$, where $A=(V(G)\setminus V(G_1))\cup\{x,y\}$.

Remark. We denote by S_{ki} the set of supporting vertices of the component K_1, $i=1,\ldots,p$. From the lemma 4.2.1 we have that $S_{Ki} \subseteq C[x,y]$ or $S_{Ki} \subseteq C[y,x]$, $i=1,\ldots,p$. This assures that $G=G_1\cup G_2$ (if $K_i \notin G_1$, then it is certain that $K_i \in G_2$, $i=1,\ldots,p$).

5. A planarity criterion for hamiltonian graphs. Any graph G=(V,E) may be represented in a plane in the following way:

(1) For each vertex x in V a point $\phi(x)$ of the plane is assigned, such that distinct points of the plane are assigned to distinct vertices of the graph.

(2) For each edge e-(x,y) a Jordan curve $\phi(e)$ of the plane is assigned, with the limits $\phi(x)$ and $\phi(y)$ such that no interior point $\phi(e)$ of the Jordan curve is the image of a vertex of

G, and two distinct Jordan curves $\phi(e)$ and $\phi(e')$, $e \neq e'$, with e,e' in E, have at most a single common point, and three distinct Jordan curves, $\phi(e)$, $\phi(e')$ and $\phi(e'')$, $e \neq e' \neq e'' \neq e$, with e, e', e" in E, does not intersect in a common interior point.

An interior point common for two Jordan curves is called intersection vertex of G. The image of G obtained in this way is called the planar representation of G. If G has a planar representation with no intersection points, G is called a planar graph, and its representation is called plane representation, or plane graph.

Let G be a planar graph. We suppose that G contains a cycle C with two overlapped C-bridges, B_1 and B_2. Then, the bridges B_1 and B_2 may not be represented either both inside C, or both outside C, without having some intersection points. Thus, it results that a bridge is represented inside C, and the other outside C. Consequently, in G the interior vertices of B_1 and the interior vertices of B_2 are separated by the cycle C, if B_1 and B_2 are regular bridges.

Remark 5.1 In a plane 2-connected graph, the border of any face is identical with its contour. A hamiltonian graph obviously is 2-connected (it has no articulation points).

Lema 5.1 Let C be a cycle of the graph G, so that O(G:C) has no edge, and $C \cup B$ is planar for each C-bridge B. Then, G has a plane representation so that C is the contour of the infinite face.

Proof. We will do an induction on the number of C-bridges. Let G be a graph which has a cycle C and only one C-bridge B, so that $C \cup B$ is planar (obviously, O(G:C) has no edge). Then, $C \cup B$ is planar and there exists a plane representation of G such that C is the contour of the infinite face.

Let us suppose that the lemma is true for any graph G and any cycle C in G which has k C-bridges, with 1 <= k <= m. Let G' a graph and C' a cycle of G' having m+1 C'-bridges with the property that $C' \cup B'$ is planar for each C'-bridge B', and O(G':C') has no edge. Since O(G':C') is not connected, from the theorem 4.2.2 there exists a 2-separation (G_1, G_2) of G' such that:

(1) $\{x,y\} = V(G_1) \cap V(G_2) \subseteq V(C')$;

(2) $C'[x,y] \subseteq G_1$ and $C'[y,x] \subseteq G_2$;

(3) neither G_1 nor G_2 is a segment of C'.

From (3) we have that G_i contains at least a C'-bridge, i=1,2. (*). As we saw in the proof of the Theorem 4.2.2, for every C'-bridge B_1 in G_1 and for every C'-bridge B_2 in G_2, we have that $S_{B1} \subseteq C'[x,y]$, $S_{B2} \subseteq C'[y,x]$.

We know that $C_1 := C'[x,y] \cup (x,y)$ is a cycle of $G_1 \cup (x,y)$, and $C_2 := C'[y,x] \cup (y,x)$ is a cycle of $G_2 \cup (y,x)$. Each C'-bridge in G' is either a C_1-bridge in $G_1 \cup (x,y)$, or a C_2-bridge in $G_2 \cup (y,x)$ (see fig. 5.1). (**)

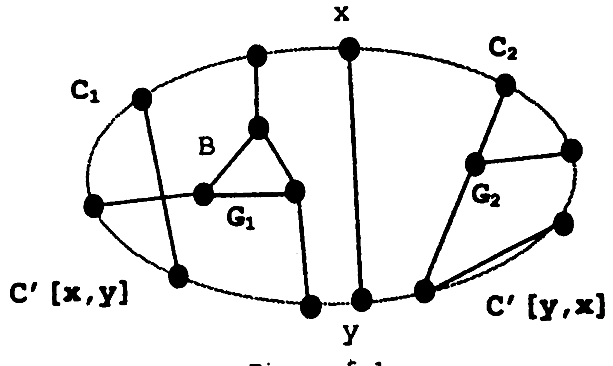

Figura 5.1

From (*) and (**) it results that the cycle C_i has at least a C_i-bridge, i=1,2. So, the cycles C_i have at least m bridges, i=1,2.

We denote $G_1^0 = G_1 \cup (x,y)$ and $G_2^0 = G_2 \cup (y,x)$. For each C_1-bridge B_1^0 from G_1^0, we have that $C' \cup B_1^0$ is planar (from the hypothesis), so $C_1 \cup B_1^0$ is planar.

Since $O(G':C')$ has no edge, we have that $O(G_1^0:C_1)$ has no edge.

Similarly, $C_2 \cup B_2^0$ is planar for each C_2-bridge B_2^0 from G_2^0, and $O(G_2^0:C_2)$ has no edge.

From the induction hypothesis, we have that G_i^0 has a plane representation so that C_i is the contour of the infinite face, i=1,2.

Let $\underline{G_i^0}$ be the plane representation of G_i^0, i=1,2.

In order to obtain a plane representation of G' with no intersection points, we represent G_2^0 in the infinite face of $\underline{G_1^0}$ and then we erase the curve $\phi(e)$ that represents the edge e=(x,y)
(see fig. 5.2).

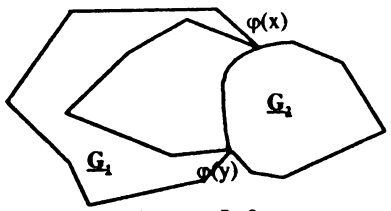

Figura 5.2.

Lemma 5.2 Let C be a cycle of the 2-connected graph G so that O(G:C) is bipartit and C∪B is planar for each C-bridge B. Then G is a planar graph.

In order to prove this lemma we mention here the following theorem, presented in [3]:

Theorem 5.1 If J and J' are closed polygonal lines from R^2, there exists a homeomorphisn $\phi: R^2 \to R^2$ so that $\phi(J) = J'$ and the image through ϕ of a segment is also a segment.

Proof of Lemma 5.2. We will use the Theorem 5.1, with the aim of placing the plane representation of some planar bridges into a well-determined region of R^2. The fact that the image through the homeomorphism is also a planar graph may be deduced from the injectivity of the homeomorphism.

We denote by $M_1 \cup M_2 = V(O(G:C))$ the parts of the bipartit graph O(G:C). Let G_i be made by the cycle C and the bridges from the set M_i, i=1,2. Since it does not exist any edge between the vertices from M_i of O(G:C), any bridge of M_i is parallel with any other bridge of M_i. So, $O(G_i:C)$ has no edge, i=1,2. From the Lemma 5.1, the graph G_1 has a plane representation $\underline{G_1}'$ so that C is the contour of the infinite face. Is known that there exists a plane representation $\underline{G}_1$ of G_1 where any edge is represented by a line segment, C being the contour of the infinite face. (*)

Let $B_1, \ldots, B_p$ the C-bridges of M_2. We know that $C \cup B_1$ is plane, and from the definition of C-bridge, and having in mind that G is 2-connected, C is the contour of a certain face in any plane representation of $C \cup B_1$. We represent B_1 in the exterior of the representation of the cycle C.

Let us suppose that we have represented B_i, $1 <= i <= k-1 <= p-1$, in the exterior of the representation of C, with no intersection points. Let us prove that $C \cup B_1 \cup B_2 \cup \ldots \cup B_k$ is planar.

Let us first show that $C \cup B_1 \cup B_2 \cup \ldots \cup B_k$ is 2-connected. We suppose that $C \cup B_1 \cup B_2 \cup \ldots \cup B_k$ is not 2-connected. There results that this graph has at least an articulation point. We have two cases:

(a) There exists an articulation point x on the cycle C. There results that there exists l, $1 <= l <= k-1$, so that B_l has only one supporting vertex, namely x. But then x is an articulation point in G, which leads to a contradiction.

(b) There exists an articulation point x in the kernels of the C-bridges B_i, $i = 1, \ldots, k-1$. From the lemma 3.1 the kernels of the C-bridges B_i, $i = 1, \ldots, k-1$ have no common vertices, and thus x is a member of only one C-bridge. By eliminating this articulation point we do not affect the other C-bridges, and the cycle C. So, it is an articulation point in G, contradiction.

Let $K=\{B_1,\ldots,B_{k-1}\}$ the set of C-bridges already represented in the plane, with no intersection vertices. Since B_k is parallel with any other bridge in K, there exists $C_k[x,y]$ (a segment of C with respect to K, with the limits x and y) containing all the supporting vertices of B_k.

From the hypothesis of the lemma we have that $C \cup B_k$ is planar. From lemma 5.1, $C \cup B_k$ has a plane representation such that C is the contour of the infinite face.

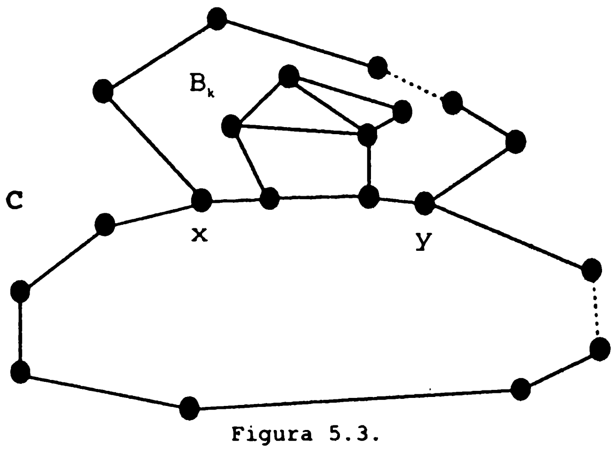

Figura 5.3.

Because $C \cup B_1 \cup \ldots \cup B_{k-1}$ is 2-connected planar, we may conclude that the border of the finite face that touches $C_k[x,y]$ is a cycle. So, from the theorem 5.1 we may represent B_k in the finite face of $C \cup B_1 \cup \ldots \cup B_{k-1}$ which touches $C_k[x,y]$, without having some intersection points (see fig. 5.3), and thus $C \cup B_1 \cup \ldots \cup B_{k-1}$ is planar.

To conclude, $C\cup B_1\cup\ldots\cup B_{k-1} = G_2$ is plane, and there exists a plane representation $\underline{G_2}$ of G_2 so that all the bridges of M_2 are represented in the exterior of the cycle C. (**)

From (*) and (**) we conclude that $G=G_1\cup G_2$ is a planar graph.

The following result is intuitively obvious:

Proposition 5.1 Let G be a planar graph with the cycle C and let B_1 and B_2 be two C-bridges. If B_1 and B_2 overlap then B_1 and B_2 cannot be represented in the same region of C.

Lemma 5.3 Let G be a 2-connected graph. If G has a subgraph H so that O(H:C) is not bipartit, then G is not planar.

Proof. By reduction to absurd. We suppose that G is planar. Then H is also planar. Because O(H:C) is not bipartit,, there exist two C-bridges both interior or exterior to C, connected by an edge of O(H:C). Thus there results that two overlapped C-bridges are represented in the same region of C, and that contradicts the proposition 5.1.

Before formulating the planarity criterion for hamiltonian graphs, we make the following remark:

Remark 5.2. We notice from fig. 5.4 that B is a planar C-bridge, but $C\cup B$ is not planar. The condition "$C\cup B$ is planar for each C-bridge B" is present in the hypotheses of the lemmas 5.1 and 5.2.

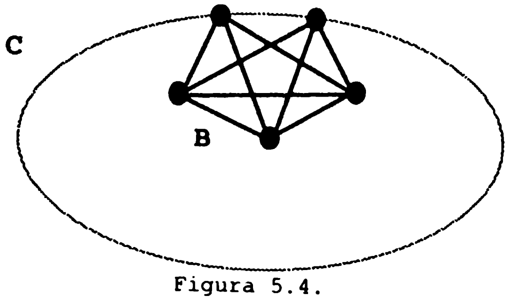

Figura 5.4.

Theorem 5.2 (Planarity criterion for hamiltonian graphs)

Let G be a hamiltonian graph and C a hamiltonian cycle of G. Then, G is planar if and only if O(G:C) is bipartit.

Proof. (==>) We prove by reduction to absurd. We suppose that O(G:C) is not bi-partit. Since G is hamiltonian, G is a graph at least 2-connected. From lemma 5.3, taking H=G there results that G is not planar, and this contradicts the hypothesis.

(<==) We have that O(G:C) is bi-partit. C being a hamiltonian cycle, all the C-bridges are diagonals of C and O(G:C) is a circle graph.

Let G' the plane representation of G, and C' the plane representation of C, so that C' is a geometrical circle and the diagonals of C are represented by line segments (see section 4). Then, C∪B is planar for each C-bridge B. From lemma 5.2 there results that G is a planar graph.

6. An algorithm for testing the planarity of a hamiltonian graph. Building of a plane representation.. The Theorem 5.2 suggests a simple algorithm for testing the planarity of a hamiltonian graph. If the given graph is planar, the algorithm will allow us to build a plane representation of it.

Algorithm 6.1:

```
S1  Determine a hamiltonian cycle C in the graph G. .
    If there are no hamiltonian cycles in G, then
            Message: "the graph G is not hamiltonian"
            STOP
    EndIf
S2  Buid the circle graph O(G:C)
S3  If O(G:C) is not bipartit then
            Message: "the graph G is not planar"
            STOP
    EndIf
S4  Let $M_1$ and $M_2$ the parts of the bipartit graph O(G:C). The vertices of G are
    represented on a circle in the plane, in their order in the cycle C. The diagonals from
    $M_1$ are represented by line segments, inside the circle. The diagonals from $M_2$ are
    represented by circle arcs, outside the circle.
```

6.1 Testing the planarity of 4-connected graphs and 3-connected graphs. We give here two known results that will help us to extend the applicability domain of the given algorithm.

Theorem 6.1.1 (W.T. Tuttle) Each 4-connected planar graph has a hamiltonian cycle.

Theorem 6.1.2 (C. Thomassen) Let G be a 3-connected planar graph with an articulation set of cardinal at most 3. Then G has a hamiltonian cycle.

From the results above, the next consequences are straightforward:

Consequence 6.1.1 Let G be a 4-connected graph. If there are no hamiltonian cycles in G, then G is not planar.

Consequence 6.1.2 Let G be a 3-connected planar graph with an articulation set of cardinal at most 3. If there are no hamiltonian cycles in G, then G is not planar.

So, we may use the algorithm 6.1 in order to test the planarity of 4-connected graphs and of 3-connected graphs with an articulation set of cardinal at nost 3. The only modification in the algorithm is the message from the step S1: "the graph G is not planar".

7. An implementation of the algorithm 6.1. Even if in this paper we only discuss about undirected graphs, in order to implement the algorithm 6.1 we chose a representation useful for oriented graphs, either.

Let G=(V,E) be an undirected graph, with |V|=n and |E|=m. We suppose that V={1,...,n} and E={1,...,m}. We establish an arbitrary orientation for the edges of the graph G and we obtain an oriented graph G'=(V,E,alpha,omega), where

alpha: E --> V, omega: E --> V and

for each e in E, alpha(e) represents the initial extremity of the arc e and omega(e) represents the final extremity of the arc e.

We define the enumeration types edge and vertex in this way:

```
edge   = -m .. m;
vertex =  1 .. n;
```

We use the array NODE: array [edge] of vertex;

in.order to store alpha(e) in NODE[-e] and omega(e) in NODE[e], for every e in E [1].

We will also use the arrays:

```
FIRST: array [vertex] of edge;
NEXT : array [edge] of edge;
```

in order to chain all the arcs incident to a vertex of the graph, the sign of an arc playing the role of direction indicator.

Remark 7.1 Any supplementary information concerning the vertices and the edges (labels, markers, and so on) may be memorized in arrays having the lenght n and m, respectively. This is one of the main advantages of representing a graph by adjacency lists, as compared with a representation by using incidence matrix. The great majority of the algorithms based on incidence matrix request that this matrix must be inspected element by element. From here it arises the imposibility to reduce the complexity of the algorithms at a value smaller that $O(n^2)$. [3]

Remark 7.2 The introduction of the graph in the memory of the computer may be simply realized by stating the extremities of each arc. Then, in order to complete the arrays FIRST and NEXT, we use the following algorithm (linear time):

```
for v:= 1 to n do.
  FIRST[v]:=0;
for e:=m downto 1 do
 begin
   NEXT[e]:=FIRST[NODE[-e]];
   FIRST[NODE[-e]]:=e;
   NEXT[-e]:=FIRST[NODE[e]];
   FIRST[NODE[e]]:=-e;
 end;
```

Considerations regarding the source text of the program

1. The following result is well known:

Let G be an undirected graph, with no loops and no multiple edges, with $|V|=n$ and $|E|=m$. If $n >= 3$ and G is planar, then $m<=3n-6$. [5]

In the function Introduction we verify if the inequality mentioned above is fulfilled. If $m > 3n-6$, we display a message concerning the nonplanarity of G, and the execution of the program stops.

2. Let G=(V,E) be a hamiltonian graph, with |V|=n and |E|=m, and be C a hamiltonian cycle of G. Then O(G:C) has exactly m-n vertices.

It is easy to show that a bipartit graph with p vertices has at most $p^2/4$ edges.

In the function GraphCon, which builds the overlap graph O(G:C), each time a new edge is added we verify that the number of edges of O(G:C) is not bypassing $[(m-n)^2/4]$. In the case of an affirmative answer, the graph O(G:C) cannot be bipartit, and thus G is not planar.

Remark 7.3 At the function GraphCerc a variable of the type set of byte has been used. This is the cause of the restriction for the order of the graph to be n <= 87. So, for the given graph with more than 87 vertices is necessary to rewrite the code of the function GraphCerc where the variable CycleHamilt is used, and, obviously, its type will be changed.

Remark 7.4 Obviously, the program may be optimized. For example, the variable succ from the procedure Colouring may be put as global variable, in order to avoid the repeated allocation in the Heap, caused by the recursive call.

REFERENCES

[1] Ebert,J. A versatile data structure for edge-oriented graph algorithms, Comm. A.C.M., vol.30(6), 1987,pp.513-519.

[2] Hopcroft,J., Tarjan,R., Efficient planarity testing, Journal A.C.M., vol.21(4),1974,pp.549-568.

[3] Thomassen,C., Kuratowski's theorem, J. Graph Theory vol.5(3),1981,pp.225-241.

[4] Voss,H.-J., Cycles and Bridges in Graphs, Kluwer Academic Publishers, 1991

STUDIA UNIV. BABEŞ-BOLYAI, MATHEMATICA, XL, 3, 1995

AN EVENT-DRIVEN SEQUENTIAL SIMULATION ALGORITHM FOR DISTRIBUTED SYSTEMS PERFORMANCE EVALUATION

Alexandru VANCEA[*] and Monica VANCEA[**]

Received: November 15, 1995
AMS subject classification: 65Y05, 68M20

REZUMAT. - Un algoritm de simulare secvenţială orientată pe evenimente pentru evaluarea performanţei sistemelor distribuite. Determinarea performanţelor de execuţie ale sistemelor distribuite reprezintă o activitate de cercetare în continuă dezvoltare, atât în ceea ce priveşte criteriile de performanţă principale care ar trebui adoptate cât şi natura algoritmilor care trebuie utilizaţi într-un astfel de studiu. În lucrare se susţine efectuarea de analize de performanţă prin aplicarea unor algoritmi care să simuleze secvenţial execuţia aplicaţiilor care se rulează într-un sistem distribuit. Lucrarea prezintă un astfel de algoritm orientat pe evenimente (event-driven) care funcţionează într-un mediu distribuit în care comunicarea se face prin transmitere de mesaje.

1. Preliminaries. The actual development of distributed software systems creates new requirements for both programming languages and operating systems. Even if distributed computing had become a very active research area in the last few years, the complexity imposed by the probability features involved in the functioning of a distributed computing system makes it very difficult to accurately determine the execution performances through a simple (static or dynamic) analysis. We find many situations in which one can derive only some lower bounds and upper bounds with a large difference between them, making (almost) meaningless the whole study.

Taking into account the inherent difficulties posed by a real-time study of a distributed execution with the actual hardware and software possibilities, we claim in this paper that sequential simulation of a distributed execution is still an important and effective method for

[*] *"Babeş-Bolyai" University, Faculty of Mathematics and Computer Science, 3400 Cluj-Napoca, Romania*

[**] *"Babeş-Bolyai" University, Faculty of Economics, 3400 Cluj-Napoca, Romania*

evaluating the performance of a distributed system.

It is very hard to find simulation models for distributed systems. There are many studies involving queue's models, but it seems finally not to be a very good choice because in distributed systems is little central control and generally the nodes have equal importance.

Sequential discrete event simulation can be made in two forms: *event driven* and *time-driven*. In the event-driven simulation events which appear in the physical system are simulated chronologically and the simulation clock is advanced after the simulation of an event to the time of the next event to be executed. In the time-driven simulation the clock advances by one tick in every step and all events scheduled at that moment in time are simulated.

The simulation considers buffered communication and takes into account only communications between adjacent nodes made through message passing. The communication between non-adjacent nodes can be transformed into communication between some pairs of adjacent nodes by the distributed path choice algorithm [Duan87].

We start our discussion from the method presented in [Ping87] which implements for every node a so called *synchronization management module* (SMM). Every module exchanges its status information by reading and writing some boolean flag variables in order to implement the synchronization of interprocessor communications. The SMMs of the different nodes have the same structure, the only difference between them being the values of some parameters which depend on the node information. In each time phase of an operation each module will perform one of the following actions:

(1) *Asking phase*: try to send a message to an adjacent node;

(2) *Responding phase*: accept messages from adjacent nodes;

(3) *Closing phase*: neither send to nor accept message from an adjacent node;

The choice is made accordingly to the values of two status variables (*buffer_full* and *send*) which indicate respectively whether the node's communication buffer is full or not and whether the current node needs or not to send a message to another node during the current fraction of time.

During distributed systems simulations the following assumptions are frequently involved [Lan84]:

(1) When a message is to be sent, the probability to be the sender node is equal for every node; also is equal the probability of every other node to be the target node;

(2) The frequency at which a node sends messages follows a negative exponent dostribution;

We mentioned that we consider only communications between adjacent nodes, so assumption (1) is changing in:

(1a). When a message is to be sent, the probability to be the sender node is equal for every node; also is equal the probability of every one of its adjacent nodes to be the target node;

2. Event-driven sequential simulation. The basic intuitive algorithm for the event-driven sequential simulation is constructed as following:

a). Input the interconnection structure of the distributed system for which the algorithm is used; set the initial status and time for each node;

b). Select the node for which the current time is the smallest;

c). Simulate the activity of this node for one or more phases and update the data structure;

d). If the number of the sent messages is not adequate go to (b), otherwise output the simulation result;

The simulation is based on the assumptions mentioned above. Two kinds of stochastic variables are involved in the simulation system: one of them follows uniform distribution and the other follows negative exponent distribution. The function $F(i+1) = (a*F(i)+b) \bmod m$ is used to generate a pseudo stochastic number, function for which the following theorem is known [Lan84]:

Theorem 1. The period of the pseudo stochastic number generator $F(i+1) = (a*F(i)+b) \bmod m$ is m, if and only if

i). $b \bmod m = 0$ and $m \bmod b = 0$;

ii). for any prime number p such that $m \bmod p = 0$, we have

$(a-1) \bmod p = 0$;

iii). if $m \bmod 4 = 0$ then $(a-1) \bmod 4 = 0$;

In general, $m = 2^k$, where k is a positive integer.

The function $\delta = \max\{F(n), 1\}/m$ is used to generate the stochastic real number which follows a uniform distribution in the (0,1) interval. From this, we obtain:

a). Negative exponent distribution with mathematical expected value $1/\lambda$: $\eta = -(\ln \delta)/\lambda$;

b). Uniform distribution in the set $\{1,2,\ldots,k\}$: $l = \lceil k*\delta \rceil$

Let λ_i be the joint probability of the following probabilities:

(1) the probability of node i to send a message to another node per phase;

(2) the probability that the communication buffer of node *i* to be full;

Let N be the number of nodes and let $R = | R_{max}/R_{min} |$, where R_{max} and R_{min} indicate the upper and lower bounds respectively of the node's speed, $0 < R_{min} \leq R_{max}$.

In order to simplify the simulation process we can assume that:

(a) the ratio of node i's speed to R_{min} is $1+i*(R-1)/N$, and it does not change in the running process;

(b) all λ_i have the same value, where $1 \leq i \leq N$;

(c) the communication buffer is large enough for never be completely full, so, no closing phases will be encountered during the simulation process;

These assumptions have no inherent effects on the validity of the simulation results.

Because the nodes speeds are different, the nodes upper bounds of one phase's length will be different. Let $upp_{l(i)}$ be node's *i* upper bound of one phase's length and status(S) and ctime(S) be the current status and the current time respectively of node S. By n(S) we denote the number of phases for which node S has successively requested to communicate.

With these considerations the general form of the simulation algorithm is as follows:

```
begin
      initialize the interconnection structure of the distributed
      system, R, λ and other functional parameters;

      while (the number of sent messages is not enough) do
            select a node S with min(ctime(S));

            case status(S)=0 :

      { 1 - simulate the environment in which the algorithm works
      according to the negative exponent distribution, compute the
      length t of the interval between ctime(S) and the moment at which
      node S wants to send the next message; according to the uniform
      distribution select the target node;}

                   ctime(S)=max(ctime(S), ctime(S)+t);

      { where t is the response time of node S's last communication }
      { 2- record related information }

                  status(S):=1;

            case status(S)=1 :  { simulate the responding phase }

                   n(S):=0; status(S):=2;

            case status(S)=2 :  { simulate the asking phase }

                   if n(S)=S then status(S):=1 else
```

```
                 begin
                  if (the communication request has been
                                       succesfully answered) then
                     begin
                       compute and record related information;
                       status(S):=0;
                     end
                  else n(S):=n(S)+1;
                 end;

               if status(S)=0 then
                ctime(S):= the time of node S at which the
                                communication is established
               else
                ctime(S):=ctime(S)+$upp_{l(S)}$;
          endcase;
     endwhile;
     output the simulation results;
end.
```

The performances and precision of such an algorithm are now under practical research on the Sun network stations at UBB Computing Centre and the results will be reported comparatively to some other related algorithms in a future paper.

REFERENCES

[Akl90] S.G.Akl - *The Design and Analysis of Parallel Algorithms*, Prentice Hall, 1989.

[Duan87] Ping Duan and Cai Xiyao - A Real-time Interprocessor Synchronization Algorithm for Communications in Distributed Computer Systems, in *Journal of Computer Science and Technology*, vol.2 (1987), no.4, pp.292-302.

[Krish89] E.V.Krishnamurthy - *Parallel Processing. Principles and Practice*, Addison-Wesley, 1989.

[Lan84] Jin Lan and Zheng Weimin - Stochastic Simulation of a Distributed Multiprocessor System, in *Chinese Journal of Computers*, vol.7 (1984), no.2, pp.100-107.

[Ping87] Duan Ping - A Real-Time Synchronization Algorithm for Interprocessor Communications Based on Duplex Systems, in *Proceedings of IEEE Asian Electronics Conference*, 1987, pp.473-475.

[Tan92] A.S.Tanenbaum - *Modern Operating Systems*, Prentice Hall, 1992.

In cel de al XL-lea an (1995) *Studia Universitatis Babeş-Bolyai* apare în următoarele serii:

matematică (trimestrial)
fizică (semestrial)
chimie (semestrial)
geologie (semestrial)
geografie (semestrial)
biologie (semestrial)
filozofie (semestrial)
sociologie-politologie (semestrial)
psihologie-pedagogie (semestrial)
ştiinţe economice (semestrial)
ştiinţe juridice (semestrial)
istorie (semestrial)
filologie (trimestrial)
teologie ortodoxă (semestrial)
educaţie fizică (semestrial)

In the XL-th year of its publication (1995) *Studia Universitatis Babeş-Bolyai* is issued in the following series:

mathematics (quarterly)
physics (semesterily)
chemistry (semesterily)
geology (semesterily)
geography (semesterily)
biology (semesterily)
philosophy (semesterily)
sociology-politology (semesterily)
psychology-pedagogy (semesterily)
economic sciences (semesterily)
juridical sciences (semesterily)
history (semesterily)
philology (quarterly)
orthodox theology (semesterily)
physical training (semesterily)

Dans sa XL-e année (1995) *Studia Universitatis Babeş-Bolyai* paraît dans les series suivantes:

mathematiques (trimestriellement)
physique (semestriellement)
chimie (semestriellement)
géologie (semestriellement)
géographie (semestriellement)
biologie (semestriellement)
philosophie (semestriellement)
sociologie-politologie (semestriellement)
psyhologie-pédagogie (semestriellement)
sciences économiques (semestriellement)
sciences juridiques (semestriellement)
histoire (semestriellement)
philologie (trimestriellement)
théologie orthodoxe (semestriellement)
éducation physique (semestriellement)