Volume 15 Number 4 November 1991
YU ISSN 0350-5596

Informatica

A Journal of Computing and Informatics

The Slovene Society INFORMATIKA
Ljubljana

Informatica

A Journal of Computing and Informatics

Subscription Information

Informatica (YU ISSN 0350 – 5596) is published four times a year in Winter, Spring, Summer and Autumn (4 issues).

The subscription price for 1991 (Volume 15) is US$ 30 for companies and US$ 15 for individuals.

Claims for missing issues will be honoured free of charge within six months after the publication date of the issue.

Printed by Tiskarna Tipa, Gosposvetska 13, Ljubljana.

Informacija za naročnike

Informatica (YU ISSN 0350 – 5596) izide štirikrat na leto, in sicer v začetku februarja, maja, avgusta in novembra.

Letna naročnina v letu 1991 (letnik 15) se oblikuje z upoštevanjem tečaja domače valute in znaša okvirno za podjetja DEM 30, za zasebnike DEM 15, za študente DEM 8, za posamezno številko pa DM 10.

Številka žiro računa: 50101 – 678 – 51841.

Zahteva za izgubljeno številko časopisa se upošteva v roku šestih mesecev od izida in je brezplačna.

Tisk: Tiskarna Tipa, Gosposvetska 13, Ljubljana.

Na podlagi mnenja Republiškega komiteja za informiranje št. 23 – 85, z dne 29. 1. 1986, je časopis **Informatica** oproščen temeljnega davka od prometa proizvodov.

Pri financiranju časopisa **Informatica** *sodeluje Republiški komite za raziskovalno dejavnost in tehnologijo, Tržaška 42, 61000 Ljubljana*

Volume 15 Number 4 November 1991
YU ISSN 0350 – 5596

Informatica

A Journal of Computing and Informatics

The Slovene Society INFORMATIKA
Ljubljana

Letnik 15 Številka 4 November 1991 YU ISSN 0350 – 5596

Informatica

Časopis za računalništvo in informatiko

V S E B I N A

Modes for Reasoning about Cosciousness	*O.B. Popov*	1
Design, Implementation and Testing of a Primal – dual Interior Point Method for Solving Linear Programs	*J. Barle* *J. Grad*	5
Pattern Recognition Using Kohonen Maps and Multi – layer Perceptrons	*Tjaša Meško*	12
Formal Informational Principles	*A.P. Železnikar*	19
Planiranje s Prologom	*M. Likar* *N. Guid*	36
Dinamično dodeljevanje procesov na polju transputerjev	*P. Zaveršek* *P. Kolbezen*	43
Ocenjevano učenje v nevronskih mrežah	*A. Dobnikar* *Jelena Ficzko* *Mira Trebar*	53
Digitalizacija in vektorizacija črtnih risb velikih dimenzij	*A. Dedić* *R. Murn* *D. Peček*	60
Uporaba predznanja v induktivnem učenju	*Nada Lavrač*	69
Možnosti za vodenje sistemov s pomočjo metod umetne inteligence	*Tanja Urbančič*	79
Povezava osebnega računalnika z robotskim krmilnikom ASEA – IRB	*Michele Leonardi* *A. Klofutar* *I. Gorkič*	84
Novice: 64 let dvoma o nastajanju UI	*A.P. Železnikar*	89
Avtorsko stvarno kazalo časopisa Informatica, letnik 15 (1991)		93

MODES FOR REASONING ABOUT CONSCIOUSNESS

INFORMATICA 4/91

Oliver B. Popov
Institute of Informatics
College of Math and Natural Sciences
University of Skopje

Keywords: intension, consciousness, knowledge representation, reasoning modes

Abstract: The ability to reason for an agent in the system S_c for the logic of consciousness LC is quite limited. To increase the power and the flexibility of the system, one needs additional modes which represent metatheorems of the original formalization. These modes are induced through plastic constraints or conditions that maintain the determination of the system. As expected, all the modes considered are questioned on the basis of their intuitive admissibility.

1. Introduction

The development of formal systems for reasoning in artificial intelligence is necessarily influenced and measured by anthropomorphic characteristics and attributes. Intuitive admissibility is one of those. This characteristic with respect to the various types of reasoning was thoroughly researched by Hintikka in epistemic and doxastic context [Hin62]. Here, some of the well-known types or modes of reasoning are examined in the system S_c for the logic of consciousness LC [Po191, Po391].

The ability to reason for an agent in the environment of the system S_c , other then the standard axioms of (PC) and the rule of inference (RE) is quite limited. It is a compromise between the flexibility and the generality of the intensional approach and the rigid power of the modal logic systems such as S4 and S5 [Che80]. In a sense, it is consistent with the heuristic paradigm in artificial intelligence.

Certain restrictions that are imposed on the class of conceptual structures $\Omega(M)$, with respect to which the system S_c is determined, increase the number of logical devices (axioms and theorems)[Hal86]. For the basic definitions and concepts refer to [Po391]. The restrictions are accomplished by defining plastic constraints on the sets of consciousness $\{\Theta(a,k)\}$. The constraints are termed plastic for the following reason: as long as a constraint holds for a particular instance of the system, that system is determined. The class of conceptual structures for which the system is determined considering given plastic constraint is denoted by $\Omega(M_r)$. The axioms of the restricted system are actually the metatheorems of the generalized system S_c.

2. Modes for Reasoning

The modes for reasoning in the primal state of knowledge, consciousness, are formalized and discussed on the basis of their intuitive admissibility.

Mode 1: (consciousness about the impossible)

The plastic constraint imposed on an arbitrary set of consciousness is:

P1: $\Lambda \varepsilon \Theta(a,k)$

where Λ denotes the empty set. For an agent to be conscious of the impossible means that it is conscious of a wff g such that g --> f & ~ f. Let the intension of f to be INT(f) and define the intension of ~ f to be INT(~ f) = K - INT(f). Denote the intension of ~ f with CINT(f), whence CINT(f) is the complement of INT(f) with respect to the set of all complexes

K. The formula g is inconsistent since the formula (f & ~ f) is canonically unsatisfiable and INT(f) $\cap$ CINT(f) = Λ.

Situations could be encountered, in the reasoning spaces encompassed by a KB (knowledge base), which may be declared to be inaccessible to any kind of activity until certain conditions are met. These situations, described by formulas of type g, may eventually lead to either a destruction of the reasoning mechanism or to violation of the entire KB. Prima facie, there is a utility in the requirement to have a reasoning agent conscious of the impossible.

While this mode of reasoning might prove useful in an epistemic context, it is inadmissible in a doxastic context. One cannot believe both propositions "There is a life on Mars" and "There is not life on Mars" in the same complex k.

Another distinction is to be made between formulas ($_a$CON(f) and $_a$CON(~ f)) and $_a$CON(f & ~ f) which represent entirely different reasoning environments. The later one requires a time resonant interval for the formulas f and ~ f, while the former refers to two disjunctive time intervals.

Finally, the behavior of the canonically unsatisfiable formula in the system PC is the argument against the inclusion of a state when someone is conscious about the impossible. Anything is derivable from a contradiction, therefore if an agent is conscious of the impossible it is conscious of everything possible. This certainly defies our intuition and it makes strong case against considering it in the system S_C.

Mode 2: (distribution of consciousness)

The process of reasoning from universal toward particular facts is accomplished by the metatheorem

(MT2) : $_a$CON(f & g) --> $_a$CON(f) & $_a$CON(g)

which is true if the plastic constraint holds: if INT(f) $\cap$ INT(g) ε Θ(a,k) then INT(f) ε Θ(a,k) and INT(g) ε Θ(a,k).

Mode 3: (collection of consciousness)

The mode that deals with the collection of consciousness is the converse of the previous one. In this case the process of reasoning goes from particular toward universal facts.

(MT3) : $_a$CON(f) & $_a$CON(g) --> $_a$CON(f & g)

is true if INT(f) ε Θ(a,k) and INT(g) ε Θ(a,k) then INT(f) $\cap$ INT(g) ε Θ(a,k).

A substitution of the "if ... then" condition in either one of the plastic constraints with the "iff" condition yields a single mode for reasoning.

Mode 4:(the omniconsciousness of truth)

If a reasoning agent is to recognize canonically true things, denoted by T, then the constraint to be met is K ε Θ(a,k) which as a result gives

(MT4) : $_a$ CON(T)

A canonically true formula is true in every complex. Hence, the intension of T must be the set of all possible complex, i. e., INT(T) = K.

Mode 5:(conscious about consciousness)

This is a case which involves iteration of epistemic operators. It is a metatheorem of positive introspection and has the form:

(MT5) : $_a$ CON(f) --> $_a$ CON(CON(f))

The metatheorem is valid when the following plastic constraint is satisfied: if INT(f) ε Θ(a,k) then there exists a set of complexes K' such that K' = {i | INT(f) ε Θ(a,i)} and K' ε Θ(a,k).

The arguments raised against the acceptance of the axiom of positive introspection in an epistemic environment are due to the (psychological) phenomena of repression and subconsciousness. It is generally considered that these phenomena are too much of speculative nature to be accepted as arguments in the context of artefact reasoning.

However, there might be some technical difficulties with the realization of either positive or negative introspection. Let δ denote the number of epistemic operators on the left side of the expression (MT5). The δ stands for a degree of reasoning about consciousness or knowledge. It can be observed that a degree of reasoning δ always induces a higher degree (δ + 1). An unlimited degree of reasoning onsequently requires infinite sets of consciousness.

Mode 6: (conscious about unconsciousness)

This mode asserts that to be unconscious implies to be conscious about something one is not unconscious off. The mode for negative introspection is a sort of linguistic caveat. The peculiar linguistic expression is formalized in the following manner.

(MT6): $_a\sim CON(f) \dashrightarrow {}_aCON(f)(\sim CON(f))$

is valid with the plastic constraint: if $INT(f) \not\in \Theta(a,k)$ then there exist a set of complexes K' such that $K' = \{i \mid INT(f) \not\in \Theta(a,i)\}$ and $K' \in \Theta(a,k)$.

It is intriguing to examine another propositional attitude, awareness, that has been introduced in a recent work on epistemology in the context of artificial intelligence and computer science. Consider the language of the logic of consciousness $\Gamma(L_c)$. Assume that $\Gamma(L_c)$ is augmented with a monadic operator 'AWE' so that ifa wff $f \in \Gamma(L_c)$ then $_aAWE(f) \in \Gamma(L_c)$. The operator of awareness 'AWE' might be viewed as a dual of the operator of consciousness 'CON' so that $V(_aAWE(f), k) = 1$ iff $CINT(f) \not\in \Theta(a,k)$, where V is the function of confirmation as defined in [Po191,Po291].

Given a complex k, a reasoning agent 'a' and a wff f, by the definition of the consciousness, for 'a' to be conscious of f means that the $INT(f) \in \Theta(a,k)$. The epistemic operator "CON' is a sort of an existential quantifier over the set of possible concepts and hence complexes for 'a'. Various concepts may exist within a given KB. Nevertheless, they do exist for 'a' if and only if the concepts are elements of the agents 'a' set of consciousness for some complex $k \in K$. When the notion of awareness is concerned , an agent is aware of a formula f if the complement of the intension of f is not an element of its set of awareness.

The previous argument does not guarantee that the intension of f is indeed in the agent's set of awareness. What is posited is illustrated in the next example. Let 'a' be conscious that the concept , i. e., the proposition "grass is green" is true in the complex k. In the case of awareness, 'a' is aware that "grass is green" in a complex k if and only if 'a' is not conscious that "grass is not green".

The problem with the inclinations such as negative introspection and even awareness is beyond intuitive admissibility. First the theoretical basis for both notions is somehow superfluous and troublesome. The reason being is that with modalities there is no 'clear cut negations' as there is in standard logical systems. The distinction between the two inclinations, consciousness and awareness, may be accepted as a distinction between definitive and possible information [Hal85].

The operator of awareness enables the alternative formulation of the negative introspection as $_aAWE(f) \dashrightarrow {}_aCON(AWE(f))$. The formulation is consistent with the idea of duality in modal and intensional theories. Absence of consciousness does not entail total ignorance about a certain concept. There is a possibility, no matter how small, that a concept is accessible.

3. Conclusion

Being quite limited in its potential to offer variety of types of reasoning, the rudimentary system for reasoning about consciousness S_c is enlarged with different modes whose intuitive admissibility is discussed. These modes of reasoning are induced through the imposition of plastic constraints. The selection of particular constraint depends on the nature of KB and the type of reasoning required for the manipulation. All of the standard axioms that are presented [Hin62] are formalized in the intensional context of the logic of consciousness LC.

References

[Che80] Chellas, B.F. Modal Logic, Cambridge University Press, 1980.

[Hal86] Halpern, J. "Reasoning about Knowledge: An Overview" in Reasoning about Knowledge ed. by Halpern, J. , (1986), Morgan Kaufmann, pp. 1-17.

[Hin62] Hintikka, J. Knowledge and Belief. Cornell University Press, 1962.

[Hin86] Hintikka, J. "The Paradigm of Epistemic Logic" in Reasoning about Knowledge, ed. by Halpern, J. , (1986), Morgan Kaufmann, pp. 63-81.

[Kap64] Kaplan, S.C. "Foundations of Intensional Logic", Doctoral Dissertation, UCLA, 1964.

[Kri63] Kripke, S. " Semantical Analysis of Modal Logic II: Non-normal Propositional Calculi" in The Theory of Models, ed. Henkin, L. and Tarski, A. , North-Holland, 1965, pp. 206-220.

[Mon74] Montague, R. Formal Philosophy Yale University Press, 1974.

[Po190] Popov,B. O. "The System of Knowledge", Informatica, Vol. 14, No. 4, October 1990, pp. 87-90.

[Po291] Popov, B. O. "An Intensional Approach towards Omniscience", International Symposium on Information Technology Proceed., Sarajevo, March 1991, pp. 235/1-235/5.

[Po391] Popov, B. O. "Consciousness as a Prerequisite for Knowledge", Informatica, Vol. 15, No. 3, August 1991, pp. 1-8.

INFORMATICA 4/91

DESIGN, IMPLEMENTATION AND TESTING OF A PRIMAL-DUAL INTERIOR POINT METHOD FOR SOLVING LINEAR PROGRAMS

Janez Barle and Janez Grad
Univerza v Ljubljani
Ekonomska fakulteta
Kardeljeva ploščad 17
61109 Ljubljana

Keywords: linear programing, primal-dual interior point algorithm, sparse Cholesky factorization

ABSTRACT: This paper describes our implementation of a primal-dual interior point algorithm for linear programming. The topics discussed include economic data structures, efficient methods for some sparse matrix operations, sparse Cholesky factorization, methods for handling dense columns and comparisons with simplex based methods. Extensive numerical results demonstrate the efficiency of the resulting algorithm as well as some problems which remain to be solved. The role of interior point based solvers in the process of solving large-scale mathematical programming models is also discussed.

SNOVANJE, IMPLEMENTACIJA IN TESTIRANJE PRIMARNO-DUALNE METODE NOTRANJE TOČKE ZA REŠEVANJE LINEARNIH PROGRAMOV: V sestavku je opisana naša implementacija primalno-dualne metode notranje točke za reševanje linearnih programov. Obravnavane so ekonomične podatkovne strukture, učinkoviti načini za izvajanje nekaterih operacij z razpršenimi matrikami, razpršeni razcep po Choleskem, metode za delo z gostimi stolpci ter primerjave z metodo simpleksov. Izčrpni rezultati numeričnega testiranja kažejo tako učinkovitost razvitih algoritmov, kot tudi nekatere probleme, ki jih je treba še razrešiti. Opisana je tudi vloga reševanja z metodami notranje točke v splošnem kontekstu modeliranja in reševanja velikih problemov matematičnega programiranja.

Introduction

The linear programming (LP) problem may be stated in the form:

$$\text{minimize (maximize) } c^T x$$
$$\text{subject to: } Ax = b, \quad l \le x \le u \qquad (1)$$

where A is a rectangular matrix, c, x, l, u, b are column vectors and the symbol ^{T} denotes transpose of a vector or matrix. Some of bounds in l and u may be infinite. The important features of typical LP constraint matrix A are its quite large dimension, sparsity and a specific structure i.e. patterns in which its nonzero elements appear. A classical way for solving the above problem is the simplex algorithm, developed by G.B. Dantzig in late 1940's. Contemporary variants of this algorithm, which are included in many commercial or university developed software packages, are tailored for solving quite large problems in a fast and reliable way. However, this established algorithm has got recently a serious competitor in the so called interior point methods. These methods became widely known after Karmarkar's publication (Karmarkar, 1984) of an algorithm that is claimed to be much faster for practical problems than the simplex method. Although these initial promises appeared to be too optimistic, Karmarkar's algorithm and other interior point methods are now regarded as a competitive methods for solving LP problems. This is particularly true when solving of some specific forms of super size problems on supercomputers is considered. Such kind of problems, which are often encountered in communication, transportation and military operations, are very sparse and usually exhibit specific and generally well-behaved block structures that can be effectively exploited. Efforts to develop software systems for solving super size LP problems with Karmarkar's algorithm proved to be very fruitful. One of the outstanding steps in this direction is AT&T's KORBX system. The system consists of both hardware, which uses parallel processing, and software which exploits the resources of this hardware (Carolan et al., 1990). However, there is also a need for exploring ability of interior point methods for solving LP on a more widely available serial computers. In the paper we present our work in this direction, which was performed on MS-DOS personal computers and VAX/VMS minicomputers.

Algorithms

Nowadays a plethora of research papers is published where different interior point methods are proposed. We have employed the variant of a primal-dual interior point method which is supposed to be among the most efficient (Lustig et al. 1989). In order to make clear differences between such kind of methods and simplex based algorithms, we first give a brief explanation of the revised simplex algorithm. The steps of this algorithm are roughly described within the following box where B denotes the basis matrix and c_B the cost vector of the basic variables. It is therefore assumed that there is a set of m basic variables, which is usually changed after each iteration in such a way that one nonbasic variable enters the basis and one basic variable leaves the basis. The informal description which follows is related to the second phase of the primal revised simplex algorithm, where the basic feasible solution is already known.

Revised-simplex-method

R1: Produce a pricing vector: $\pi = c_B^T B^{-1}$ (BTRAN).

R2: Select the entering variable x_s (column $u = A_{*s}$) according to a given pricing strategy. If no entering variable is found, terminate (solution is optimal).

R3: Update the entering column: $v = B^{-1}u$ (FTRAN).

R4: Determine the leaving variable. If none is found, terminate (problem is unbounded).

R5: Update the basis matrix representation; refactorize if necessary. Go to R1.

It must be noted however that there is not yet general agreement about what are the best algorithms in detail, and how they should be implemented in the most efficient way. In general, number crunching operations are concentrated within the steps R1 and R3 where two systems of linear equations have to be solved (these operations are often refered to as BTRAN and FTRAN). It is very important that after basis change updating of basis matrix is possible without performing full factorization, which has to be done only periodically. Other steps, particularly R2 and R4, deal mainly with logic decision and "book-keeping" problems. It is also obvious that a rather sophisticated data structures must be employed in order to exploit sparsity (Duff et al., 1989).

Interior point methods differ considerably from the simplex method. Primal-dual interior point method, which we have decided to implement, requires LP problem being formulated in the following form:

$$\begin{aligned} &\text{minimize} && c^T x \\ &\text{subject to:} && Ax = b,\quad x + s = u,\quad x \geq 0,\ s \geq 0 \end{aligned} \tag{2}$$

with the associated dual

$$\begin{aligned} &\text{maximize} && b^T y - u^T w \\ &\text{subject to:} && A^T y - w + z = c,\quad w \geq 0,\ z \geq 0 \end{aligned} \tag{3}$$

Fortunately, formulation (2) can be derived in a straightforward way from formulation (1). An outline of the algorithm is sketched within the following box, where X, S, W and Z are diagonal matrices with diagonal elements equal to the components of corresponding vectors x, s, w and z. Φ is the user supplied constant which is usually computed by using the following formula:

$$\Phi = \Phi(n) = \begin{cases} n^2 & n \leq 5000 \\ n\sqrt{n} & n > 5000 \end{cases}$$

and $M = \tau\Phi(n) * \max\{\|c\|_\infty, \|b\|_\infty\}$ where τ is a scalar multiplier which is used to allow variations of the initial μ. Furthermore, $d_D = A^T y^0 + z^0 - c$, where y^0 and z^0 are initial y and z, and $e = (1,1,\ldots,1)$. α_d, α_p are some appropriate step lengths in the primal and dual spaces respectively. These step lengths must be chosen in a way which ensures nonnegativity of variables x, s, z and w, for example:

$$\alpha_p = \alpha_0 * \min\{\min_j\{x_j/-\delta x_j,\ \delta x_j<0\},\ \min_j\{s_j/\delta x_j,\ \delta x_j>0\}\}$$

$$\alpha_d = \alpha_0 * \min\{\min_j\{z_j/-\delta z_j,\ \delta z_j<0\},\ \min_j\{w_j/-\delta w_j,\ \delta w_j<0\}\}$$

where $0<\alpha_0<1$ is the user supplied parameter which is usually set to be equal 0.9995.

Initial feasibility of the problem is formally assured by adding column $Ax_0 - b$ and row d_D to the matrix A, together with x_a and $y_a (=-z_b)$, which are corresponding primal and dual variable with initial value 1. In order to achieve feasibility their values must decrease to 0.

Primal-dual-interior-point-method

K1: Compute $\mu = (c^T x + u^T w - b^T y + M(x_a - y_a))/\Phi$

K2: Compute $\theta = (S^{-1}W + X^{-1}Z)^{-1}$

K3: Compute positive definite matrix $A\theta A^T$.

K4: Perform Cholesky factorization of the $A\theta A^T$

K5: Compute $\sigma(\mu) = \mu(S^{-1} - X^{-1})e - (W - Z)e$

K6: Compute
$$\begin{aligned} \delta y &= -(A\theta A^T)^{-1}(A\theta(\sigma(\mu) + z_b d_b) + (Ax - b)) \\ \delta x &= \theta(A^T\delta y + \sigma(\mu) - z_b d_b) \\ \delta s &= -\delta x \\ \delta z &= -\mu X^{-1}e + Ze - X^{-1}Z\delta x \\ \delta w &= -\mu S^{-1}e + We - S^{-1}W\delta x \end{aligned}$$

K7: Update
$$\begin{aligned} y_{new} &= y_{old} - \alpha_d \delta y \\ x_{new} &= x_{old} - \alpha_p \delta x \\ s_{new} &= s_{old} - \alpha_p \delta s \\ z_{new} &= z_{old} - \alpha_d \delta z \\ w_{new} &= w_{old} - \alpha_d \delta w \end{aligned}$$

K8: If relative duality gap satisfies relation:

$$\frac{c^T x + u^T w - b^T y}{1 + |u^T w - b^T y|} < \epsilon$$

where ϵ is user supplied constant, terminate (solution is optimal). Otherwise go to K1.

It was also assumed that the initial interior solution is supplied by the user. For example, it is possible to choose $x^0 = z^0 = \min\{e, u/2\}$ and $y_0 = 0$ (Choi et al., 1990). In general, interior point methods are not very sensible to the choice of the initial solution.

The above description is based on two papers (Lustig et al. 1989, Choi et al. 1990) where algorithmic aspects of the modularized fortran code OB1 (Optimization with Barriers 1) were described. Our intention was to develop our own code based on mentioned papers and standard methods for computing sparse Cholesky factorization (George, Liu, 1981). However, some of the implementation details are different, for example:

a) In our implementation column $Ax_0 - b$ and row d_D are computed at each iteration, rather than only for the initial solution. x_a and y_a are defined as ratios between current and initial ∞ norms of $Ax_0 - b$ and d_D. Furthermore, x_a and y_a retain some small value even in the case if their computed vsalue is zero. Such approach enable us to save some storage space and also, according to our experience, to improve accuracy of the computed solution.

b) Sometimes it is impossible for relative duality gap to reach prescribed ϵ on step K8. In such cases we terminate algorithm when relative difference between two subsequent objective values become smaller than the prescribed constant, which is usuallly set to be $0.1*\epsilon$.

On the whole, published description of the algorithm is good enough to enable everyone to create, possibly after some experimental investigation, workable implementation of a primal dual interior point method. Evidently the algorithm consists mostly of floating point computations and consequently fortran is an obvious choice of implementation language. Some features of the interior point methods that make them so much different from the simplex method are obvious:

1) There is no partitioning into basic and nonbasic variables. This means that, in principle, all variables and constraints are handled in equal way during the solution process.

2) Each iteration requires computationally expensive factorisation of positive definite matrix or solution of the least squares problem.

3) Solution vector x is always an interior point of the solution polytope.

Feature 1) has a far-reaching consequences. It can be a means for avoiding potential combinatorial problems arising in the movement from one basis to another which is typical for simplex method. On the other hand such approach may degrade computational speed and stability.

The main computational problem of the interior point methods is inversion of matrix $A\Theta A^T$ or solution of the corresponding linear least squares problem. This is usually done by computing sparse Cholesky factorization of $A\Theta A^T$. In order to understand methods for doing this, one must be acquainted with the methods for storing sparse matrices. In the next section the methods for storing sparse matrices which were applied in our implementation of primal-dual interior point methods are briefly described.

Data structures and implementation issues

Exploitation of sparsity is based on the fact that only nonzero elements of sparse matrix (or vector) must be stored, together with information about their position within matrix (vector). In the case of LP input data (A, b, c, u) within the framework of interior point methods, we have employed the following data structures:

1) Righthand-side vector b is stored as a dense vector.

2) Constraint matrix A is stored using three one dimensional vectors (XA,IA,CP) where

   XA = vector of nonzero values $A_{i,j}$ which are sorted by columns and (secondary) by row indices within a particular column, both in increased order.
   IA = vector of row indices of nonzero elements, which are sorted in a same manner as XA.
   CP = vector of column pointers which consists of locations where the representation of columns begins in XA and IA. For example, elements of column i are all in locations from CP(I) to CP(I+1)-1.

3) Nonzero elements of c are stored (formally) as n+1. column of A. Therefore they are stored between locations CP(N+1) and CP(N+2)-1 in XA (values) and IA (indices).

4) Noninfinite elements of u are stored (formally) as n+2. column of A. Therefore they are stored between locations CP(N+2) and CP(N+3)-1 in XA (values) and IA (indices).

Obviously, it is also necessary to store the matrix $A\Theta A^T$ and its triangular factor L, in the case if Cholesky factorization is used within the solution process. These matrices can be stored using the same storage locations. The amount of this storage is determined by fill-in which generally can not be avoided during the Cholesky factorization of $A\Theta A^T$. It is therefore advisable to try to minimize fill-in by appropriate reordering of rows and columns of $A\Theta A^T$. Ordering algorithms are essentially graph techniques for obtaining appropriate numbering of the graph nodes. In our case nonzero structure of $A\Theta A^T$ represents an undirected graph G(X,E) with m nodes. The adjacency list of node $x \in X$ is a list containing all nodes adjacent to x, which is represented by indices of nondiagonal nonzero elements of corresponding column of $A\Theta A^T$. The implementation of described structure is done by storing the adjacency lists sequentially in integer array ADJNCY along with an index vector XADJ of length M+1 containing pointers to the beginning of the lists in ADJNCY. The extra entry XADJ(M+1) points to the next available location in ADJNCY (George, Liu, 1981). These arrays are input data for ordering algorithms which can be generally divided in two groups:

a) reorderings which try to minimize number of nonzero elements (and therefore fill-in) in triangular factor L. Although it is known to be NP-complete problem (Duff et al. 1989) several reasonable good heuristics exist. One of them is the minimum degree algorithm. The name of this algorithm is derived from its graph theoretic interpretation: in the graph associated with a symmetric sparse matrix, this strategy corresponds to choosing that node for the next elimination which has the least edges connected to it.

b) reorderings which try to permute $A\Theta A^T$ and triangular factor L into some particular desirable form. This can be for example so-called envelope or profile form. The most known algorithm of this type is the Reverse Cuthill-McKee algorithm. The objective of such kind of algorithms is to reorder the rows and columns of the matrix so that the nonzeroes in the obtained matrix are clustered near the main diagonal since this property is retained in the corresponding Cholesky factor L. Such a cluster is called the profile or envelope and is defined to contain also all zero elements between the diagonal and the last nonzero element in the row or column. The problem of minimizing the envelope size of a matrix is proven to be NP-complete (Billionet, Breteau, 1989) and consequently the Reverse Cuthill-McKee algorithm is only one among heuristic procedures for doing this.

We have implemented both the minimum degree and the Reverse Cuthill-McKee algorithm within our LP package. In order to give some insight into these methods, we shall show how they perform on the smallest example from the NETLIB library (Gay, 1985), which is known under the name AFIRO. This is the problem with constraint matrix having 27 rows and 51 columns which contain all together 102 nonzero elements. Its corresponding $A\Theta A^T$ matrix has the structure as in the following picture, where only the upper triangular part is reproduced:

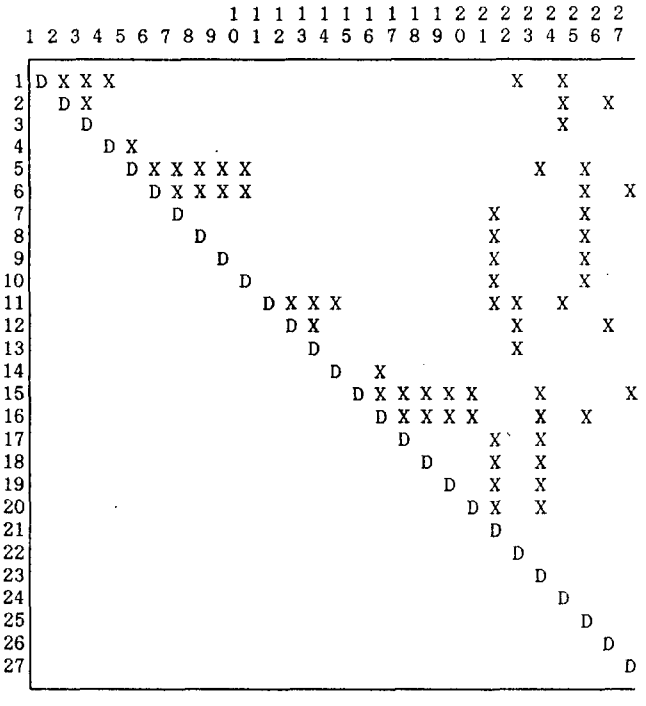


Figure 1. AFIRO - structure of the upper part of $A\Theta A^T$

Nondiagonal and diagonal nonzero elements are presented using symbols X and D respectively. It is obvious that, at least in this case, matrix $A\Theta A^T$ is not as sparse as matrix A itself. Moreover, number of its nonzeroes may substantially increase during the subsequent Cholesky factorization. The following picture displays how this situtation is controlled by applying the minimum degree algorithm. The produced ordering (permutation of rows and columns) is: (4,14,26,27,13,22,12,3,24,2,1,11,10,17, 18,19,20,23,16,15,9,8,25,21,6,7,5)

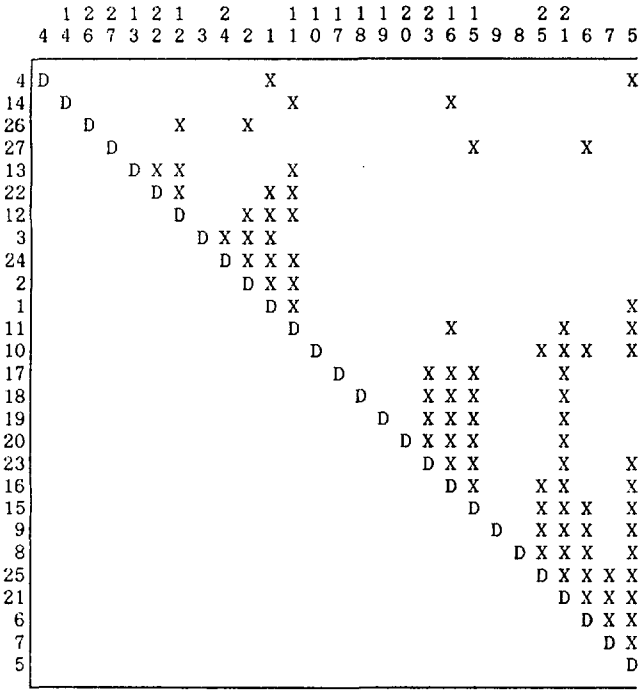


Figure 2. AFIRO - structure of the Cholesky factor

The above structure is determined not by actual numerical factorization but by simulation of it, or so called symbolic factorization. The advantage of this approach is that the data structures are set up once for all, as the structure of the matrix does not change from one iteration to another. Obviously, at each iteration it is necessary to perform numerical factorization since $A\Theta A^T$ is changing along with diagonal matrix Θ. The data structures returned by the symbolic factorization can be presented in a sparse storage scheme known as the compressed storage scheme of Sherman, cited in (George, Liu, 1981). The scheme has a storage array LNZ which will contain all nonzero entries in the nondiagonal part of Cholesky factor L column-wise (or, which are the same numbers, in the L^T row-wise), an INTEGER vector NZSUB which will hold the row subscripts of the nonzeroes, and an index vector XLNZ whose entries are pointers to the beginning of nonzeroes in each column in LNZ. The diagonal elements are stored separately in vector DIAG. In addition, an index vector XNZSUB is also used to hold pointers to the start of row subscripts in NZSUB for each column. This is the consequence of the key idea of the Sherman's compressed scheme: some of indices from NZSUB can be used for presenting nonzero patterns of two or even more columns. For example, it is applicable when columns 17,18,19,20 from the Figure 2 are considered.

Described data structure is filled et each iteration after computation of $A\Theta A^T$ is performed. The subsequent Cholesky factorization is performed using the same data structure. An interesting question is what is the best way to compute $A\Theta A^T$. At first sight row-oriented data structure for matrix A seems to be more practical and efficient when computation of $A\Theta A^T$ is considered. However, theoretical arguments (Duff et al. 1989) and practical testing convinced us that column-wise storage of A is much more efficient. The point is in avoiding operations with zero components of A. Perhaps this can be best explained with our fortran code for computating $A\Theta A^T$, which follows.

```
C*************************************************************
C     At this moment LNZ and working vector Z must be
C     initialised to zero values.
      DO 500 ICOL=1,N
         ISTART = CP(ICOL)
         ISTOP = CP(ICOL+1) - 1
         DO 100 I=ISTART,ISTOP
            J = IA(I)
            Z(J) = XA(I)*THETA(ICOL)
            DIAG(J) = DIAG(J) + Z(J)*XA(I)
  100    CONTINUE
         DO 200 I=ISTART,ISTOP-1
            IROW = IA(I)
            Z0 = Z(IROW)
            L = XLNZ(IROW)
            KSUB = XNZSUB(IROW)
            JBEGIN = I + 1
            DO 150 J=JBEGIN,ISTOP
  125          CONTINUE
               IF (NZSUB(KSUB) .EQ. IA(J)) THEN
                  LNZ(L) = LNZ(L) + Z0*XA(J)
               ELSE
                  L = L + 1
                  KSUB = KSUB + 1
                  GOTO 125
               ENDIF
  150       CONTINUE
  200    CONTINUE
         DO 300 I=ISTART,ISTOP
            Z(IA(I)) = 0.0
  300    CONTINUE
  500 CONTINUE
C*************************************************************
```

Another ordering algorithm which we have implemented is the Reverse Cuthill-McKee algorithm. Its performance on our test problem is presented on the following picture.

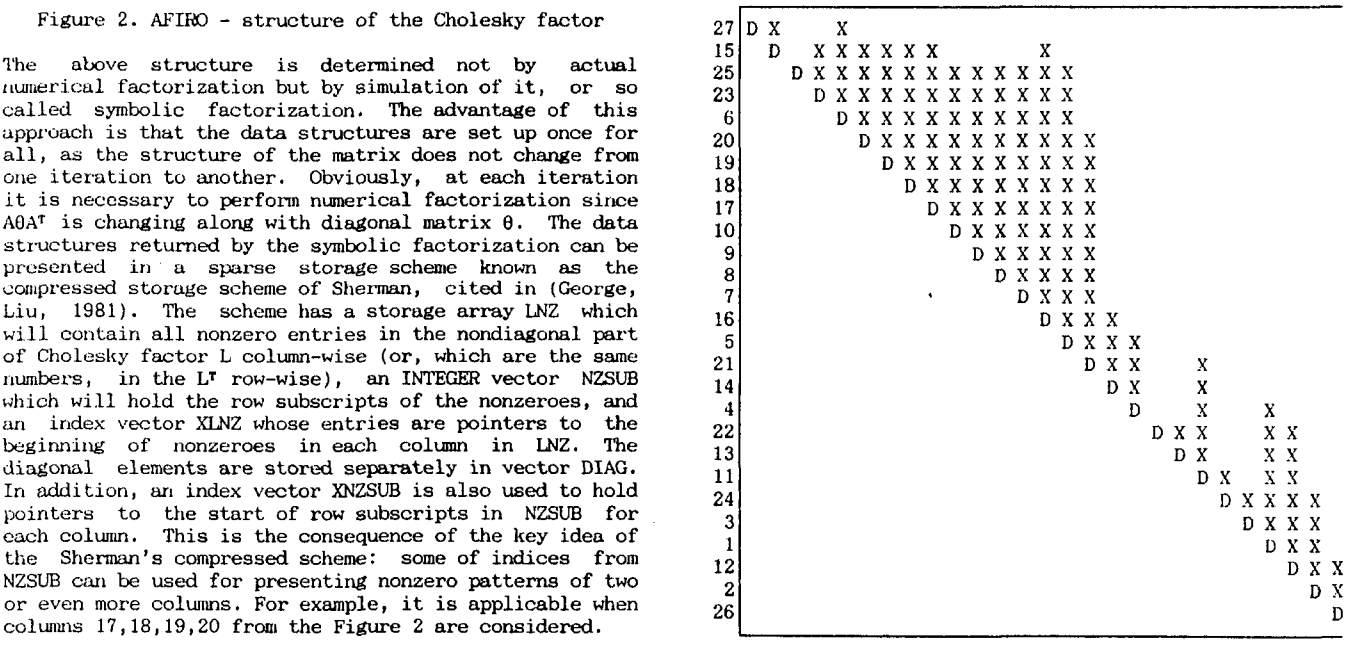


Figure 3. AFIRO - Envelope after RCM algorithm

The storage scheme for envelope methods has a main storage array ENV which will contain all entries (zeroes as well as nonzeroes) between the first nonzero entry in row of L (or column of L^T) and the diagonal, an index vector XENV whose entries are pointers to the beginning of nonzeroes in each row of L, and a vector DIAG where diagonal entries are stored.

Our practical experience with the Reverse Cuthill-McKee algorithm was quite disappointing. Its corresponding storage scheme is usually not as economical as those produced by the minimum degree algorithm. The results concerning speed of the numerical factorization were even less competitive. However, it would be interesting to try other profile methods. Some of them produce more economical storage scheme than the Reverse Cuthill-McKee algoritm (Billionet, Breteau, 1989). There are also some other efficient ordering algorithms, for example the nested dissection ordering algorithm (George, Liu, 1981). Their quantitative and qualitative comparisons may be an interesting topics for further research.

Handling of dense columns

The problem is what to do if one or more columns of A are dense vectors. In such cases computation of $A\Theta A^T$ leads to a very dense matrix and therefore should be avoided, if it is possible. This situation can be overcome by first dividing columns of A into dense and sparse submatrices:

$$A = [A_s ; A_d] \quad \text{and consequently} \quad \theta = [\theta_s ; \theta_d]$$

and then performing incomplete Cholesky factorization:

$$LL^T = A_s \theta_s$$

After this is done several approaches are possible. One among them is to use the incomplete Cholesky factors as preconditioner for a so called conjugate gradient algorithm (Adler et al. 1989). Another way is to use a method which deals with "dense windows" (Andersen et al. 1990). This method solves equation $A\Theta A^T y = q$ by performing the following operations:

Compute $V = A_d \theta_d$

Set up

$$\begin{bmatrix} LL^T & -V \\ V^T & I \end{bmatrix} \begin{bmatrix} y \\ \delta \end{bmatrix} = \begin{bmatrix} q \\ 0 \end{bmatrix}$$

Solve for δ by dense Cholesky factorization:

$$[I + V^T (LL^T)^{-1} V]\delta = -V^T (LL^T)^{-1} q$$

Compute $y = (LL^T)^{-1}(q+V\delta)$

It is obvious that typically there are only a few dense columns in A and therefore computing and storing dense Cholesky factorization is a trivial task.

Unfortunately, it was pointed out (Lustig et al., 1989) that removing dense columns can severely exacerbate the problem of ill-conditioning on badly conditioned problems. For this reason a search of reliable methods for handling dense columns remains an open research problem. One among possible approaches is to exploit fact that LP problem generally can be formulated in many different but equivalent ways. For example, it is possible to split dense column into two or more new columns which may result in sparser $A\Theta A^T$ (Gondzio, 1991). Just to give impression about that approach, we note that the following two matrices represent two different but equivalent formulations:

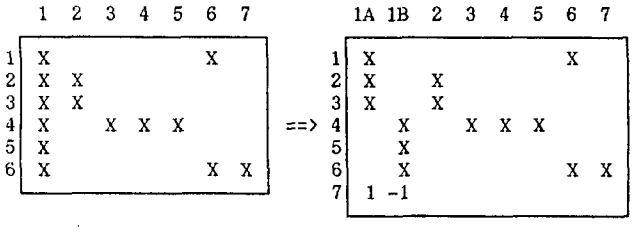


Figure 4. Column splitting

The first formulation will lead to completely dense $A\Theta A^T$, while this is not so for the second one.

Computational results

In this section, we report the computational results of running our implementation of a primal-dual interior point method on a set of LP test problems available through NETLIB (Gay, 1985). NETLIB is a system designed to provide efficient distribution of public domain software to the scientific community through different computer networks. We considered in our tests all 53 problems which are currently available to us. However, 3 of them (CZPROB, 80BAU3 and PILOTS) we were not able to read from the input file due to insufficient generality of our subroutine for reading LP data written in MPS format. Therefore we have used only 50 problems in our tests. Some quantitative data related to their size and storage consumption (when the minimum degree algorithm is used) are presented in the following table.

Problem:	No:	m:	n:	nz(A):	nz(AA^T):	nz(L):	nz(ind):
25FV47	45	820	1876	10705	11894	34590	8925
ADLITTLE	2	56	138	424	384	355	171
AFIRO	1	27	51	102	90	80	46
BANDM	19	305	472	2494	3724	4355	1913
BEACONFD	24	173	295	3408	2842	2727	1783
BORE3D	9	233	333	1446	2424	2860	1331
BRANDY	14	193	303	2202	2734	3236	1306
CAPRI	11	271	480	1933	3112	5569	2628
E226	20	223	472	2768	2823	3416	1390
ETAMACRO	16	400	734	2188	2771	11186	4016
FFFFF800	34	524	1028	6401	10615	18520	6601
GANGES	35	1309	1706	6937	8965	29359	7080
GFRDPNC	23	616	1160	2445	1451	1537	1270
GROW15	32	300	645	5620	3430	5790	5372
GROW22	38	440	946	8252	5040	8590	8039
GROW7	18	140	301	2612	1590	2590	2324
ISRAEL	15	174	316	2443	11227	11259	1373
NESM	47	662	2930	13260	4743	21283	8604
PILOT4	29	410	1181	7242	6743	14273	6781
PILOTJA	49	924	2044	13339	14174	50793	12560
PILOTWE	40	722	2930	9537	5547	15422	6109
RECIPE	6	87	178	652	582	584	153
SC205	3	205	317	665	656	986	701
SCAGR25	12	471	671	1725	2393	2510	1521
SCAGR7	4	129	185	465	629	636	417
SCFXM1	17	330	600	2732	3233	4400	1797
SCFXM2	30	660	1200	5469	6486	8977	3673
SCFXM3	36	990	1800	8206	9739	13631	5556
SCORPION	10	388	466	1534	2101	2086	937
SCRS8	26	490	1275	3288	2198	6117	2791
SCSD1	22	77	760	2388	1133	1315	464
SCSD6	31	147	1350	4316	2099	2398	960
SCSD8	46	397	2750	8584	4280	5482	1682
SCTAP1	13	300	660	1872	1686	2261	1430
SCTAP2	37	1090	2500	7334	6595	13729	6719
SCTAP3	43	1480	3340	9734	8866	17156	8636
SEBA	27	515	1036	4360	51915	53695	6176
SHARE1B	8	117	253	1179	1001	1345	526
SHARE2B	5	96	162	777	871	925	350
SHELL	28	536	1527	3058	1991	3556	2099
SHIP04L	39	360	2166	6380	4588	4428	1608
SHIP04S	33	360	1506	4400	3272	3252	1198
SHIP08L	50	712	4363	12882	9224	8948	3224
SHIP08S	42	712	2467	7194	5440	5464	2160
SHIP12L	51	1042	5533	16276	11715	11193	4742
SHIP12S	44	1042	2869	8284	6387	6289	2063
SIERRA	41	1222	2715	7951	6118	11665	5742
STAIR	25	356	538	3831	6653	15281	6263
STANDATA	21	359	1258	3173	1758	2726	1505
VTPBASE	7	198	329	945	1773	2217	974

Table 1. Quantitative data about LP problems

Numbers in the second column are related to the sequence of problems ordered by the number of nonzeroes. The last two columns show numbers of used entries in LNZ and NZSUB resceptively. On the one hand these data show that usage of compressed storage scheme is very efficient, but on the other hand indicate that density of triangular Cholesky factor can be a serious problem. In general, storage consumption of our implementation can be estimated using the following approximate formula:

real numbers (REAL*8)...: 6n + 4m + nz(A+c+u) + nz(L)
integers.....: n + 4m + nz(A+c+u) + nz(ind)

where nz(A+c+u) is the number of nontrivial elements in A,c and u together. It must be noted that interior point methods generally consume more storage than simplex based methods. Nevertheless, use of the above formula

shows that about 40 out of 50 problems can be solved on a standard PC with no more than 640 KB memory.
Our computational testing was performed on a VAX8550 computer. The operating system was VMS, version 5.2, and the VMS FORTRAN compiler, version 5.2, was used with the default options. The algorithm was used with default parameters α_0=0.9995, τ=0.1 and ϵ=10^{-8}. Computed optimal objective value and number of iterations were compared with those obtained using OB1 (Lustig et al. 1989).

Problem:	Iter. (Lustig)	Computed opt. value	Relative d. gap	Opt. value Lustig et al.
25FV47	48 (48)	5501.8459	0.321e-06	5501.8459
ADLITTLE	21 (17)	225494.96	0.447e-07	225494.96
AFIRO	14 (13)	-464.75314	0.249e-08	-464.75314
BANDM	31 (28)	-158.62802	0.579e-07	-158.62802
BEACONFD	25 (21)	33592.486	0.427e-07	33592.486
BORE3D	28 (25)	1373.0804	0.184e-08	1373.0804
BRANDY	34 (27)	1518.5100	0.412e-07	1518.5099
CAPRI	40 (37)	2690.0165	0.443e-05	2690.0127
E226	34 (31)	-18.751929	0.371e-07	-18.751929
ETAMACRO	51 (52)	-755.71523	0.132e-07	-755.71523
FFFFF800	66 (59)	555679.61	0.869e-06	555679.56
GANGES	41 (33)	-109585.74	0.670e-08	-109585.74
GFRDPNC	30 (26)	6902236.0	0.896e-07	6902236.0
GROW15	34 (25)	-1.0687094e+8	0.105e-07	-1.0687094e+8
GROW22	32 (30)	-1.6083431e+8	0.347e-06	-1.6083434e+8
GROW7	30 (22)	-47787812.	0.519e-08	-47787812.
ISRAEL	47 (47)	-896644.82	0.593e-07	-896644.82
NESM	70 (66)	14076037.	0.218e-04	14076036.
PILOT4	58 (56)	-2581.1378	0.779e-06	-2581.1405
PILOTJA	67 (67)	-6113.1349	0.155e-06	-6113.1365
PILOTWE	74 (71)	-2.7201067e+6	0.728e-06	-2.7201075e+6
RECIPE	18 (16)	-266.61600	0.867e-08	-266.61600
SC205	22 (16)	-52.202061	0.785e-07	-52.202061
SCAGR25	28 (24)	-14753433.	0.132e-06	-14753433.
SCAGR7	22 (22)	-2331389.8	0.316e-07	-2331389.8
SCFXM1	38 (31)	18416.759	0.192e-06	18416.759
SCFXM2	38 (37)	36660.263	0.149e-06	36660.262
SCFXM3	41 (39)	54901.255	0.194e-06	54901.255
SCORPION	21 (18)	1878.1250	0.141e-06	1878.1248
SCRS8	51 (50)	904.29696	0.453e-07	904.29695
SCSD1	14 (12)	8.6666667	0.320e-07	8.6666667
SCSD6	15 (15)	50.500000	0.638e-08	50.500000
SCSD8	14 (15)	905.00001	0.998e-08	905.00000
SCTAP1	22 (22)	1412.2500	0.643e-10	1412.2500
SCTAP2	23 (23)	1724.8071	0.809e-08	1724.8071
SCTAP3	27 (26)	1424.0000	0.150e-08	1424.0000
SEBA	32 (29)	15711.600	0.470e-07	15711.600
SHARE1B	42 (40)	-76589.318	0.254e-07	-76589.319
SHARE2B	20 (17)	-415.73224	0.778e-07	-415.73224
SHELL	39 (37)	1.2088253e+9	0.116e-06	1.2088253e+9
SHIP04L	26 (22)	1793324.5	0.161e-06	1793324.5
SHIP04S	22 (21)	1798714.7	0.124e-06	1798714.7
SHIP08L	26 (24)	1909055.2	0.111e-06	1909055.2
SHIP08S	25 (23)	1920098.2	0.856e-07	1920098.2
SHIP12L	29 (27)	1470187.9	0.139e-06	1470187.9
SHIP12S	29 (27)	1489236.1	0.161e-06	1489236.1
SIERRA	30 (26)	15394362.	0.140e-06	15394362.
STAIR	27 (25)	-251.26695	0.187e-06	-251.26695
STANDATA	34 (28)	1257.6995	0.295e-07	1257.6995
VTPBASE	28 (24)	129831.46	0.414e-08	129831.46

Table 2. Computational results

Our testing was therefore successful in a sense that we have solved all 50 problems with reasonable accuracy. However, we have experienced numerical troubles (such as floating overflow or underflow) on some problems:

1) To solve problem SCFXM1 we had to use α_0=0.95.
2) To solve problems PILOT4, PILOTWE and PILOTJA, which are known to be very ill-conditioned, we had to use α_0=0.95 again and also to change initial τ to value τ=0.001.

Although we have eventually solved above problems after some experimentation with algorithm parameters, a sound solution would be to use some kind of scaling. This means that instead of $A\Theta A^T$, in some cases it is better to work with the matrix $RA\Theta A^TR$, where R is a suitable chosen diagonal matrix.

On the whole, we were not able to obtain such quality of solution as it was reported for OB1, neither in terms of number of iterations, nor in terms of obtained relative duality gap. This is easily explainable by the fact that many fine details and important options, which are present in OB1, are not yet implemented in our code.

We have performed also some other types of computational testing, just to check validity of some assumptions about interior point methods. For example:
- A good simplex code usually outperforms interior point methods on small problems.
- Some problems, which are supposed to be difficult for simplex based methods, can be easily (and much faster) solved if some interior point method is used. For example, this is the case with problem 25FV47.
- Computational work within a step of interior point methods is dominated by sparse Cholesky factorization of $A\Theta A^T$. Its share on bigger problems is 60% to 90% of overall time.

System design and implementation

Implementations of interior point method were done in a form of highly portable fortran modules LPENV and LPMDG. The former uses the Reverse Cuthill-McKee algorithm, the latter uses the minimum degree algorithm. They are still in the phase of testing and development, for which we are using mainly VAX/VMS computer system, although the same code is running also on the PC. Our ultimate goal is to create reliable, portable and user-friendly LP software package based on the interior point algorithms, which is to be called LPINT. In our opinion such kind of system must contain portable full screen user interface and also subroutines for graphical display of different LP matrices (Alvarado, 1990). It is also very important to allow user to include his/her subroutines into the package. Data exchange between a user program and LPINT may be done with the usage of some type of communication region (CR). An overall LPINT system architecture can be ilustrated with the following picture:

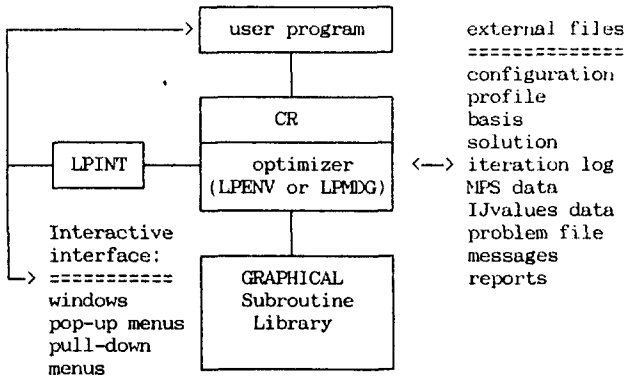


Figure 5. LPINT system architecture

Conclusions

Generally we can say that the interior point methods are getting more and more reliable and sophisticated as well. Moreover, interior point methods had greatly influenced algorithmic and experimental work in the field of linear programming. However, it is not likely that interior point methods can completely replace the simplex method in future. In our opinion reasons for this are mainly in simplex method ability to produce optimal basic solution. The knowledge of basic status of variables is very important, especially when postoptimal analysis or solution of mixed integer programs are considered. On the other hand, when one wish to solve big LP problem for the first time, it is advisable to start with some interior point based package. It is a

fast, reliable and robust way to obtain the first information about LP model.

A primal-dual interior point method can be implemented in a rather easy and straightforward way. This fact, as well as method's efficiency and mathematical elegance, can be a big step toward deeper understanding of interior point methods in general.

References

1. Adler, I., Karmarkar, N., Resende. M. and Veiga, G. (1989), "An Implementation of Karmarkar's Algorithm for Linear Programming", Mathematical Programming, Vol. 44, pp. 297-335.

2. Alvarado, F.L. (1990), "Manipulation and visualization of sparse matrices", ORSA Journal on Computing, Vol. 2, pp. 187-207.

3. Andersen, J., Levkovitz, R., Mitra, G. and Tamiz, M. (1990), "Adopting Interior Point Algorithm for the Solution of LPs on Serial, Coarse Grain Parallel Computers". Presented at the International Symposium on Interior Point Methods for Linear Programming: Theory and Practice, January 18-19, 1990, Europe Hotel, Scheveningen, Netherlands.

4. Billionnet, A. and Breteau, J.F. (1990), "A Comparison of Three Algorithms for Reducing the Profile of a Sparse Matrix", Recherche Opérationnelle, Vol. 23, pp. 289-302.

5. Carolan, W., Hill, J., Kennington, J., Niemi, S. and Witchman S. (1990), "An Empirical Evaluation of the KORBX Algorithms for Military Airlift Applications", Operations Research, Vol. 38, pp. 240-248.

6. Choi, C., Monma, C.L. and Shanno, D.F. (1990), "Further Development of a Primal-Dual Interior Point Method", ORSA Journal on Computing, Vol. 2, pp. 304-311.

7. Duff, I.S., Erisman, A.M. and Reid, J.K. (1989), Direct Methods for Sparse Matrices, Clarendon Press, Oxford.

8. Gay, D.M. (1985), "Electronic Mail Distribution of Linear Programming Test Problems", Mathematical Programming Society COAL Newsletter, Vol. 13, pp. 10-12.

9. George, J.A. and Liu, J.W. (1981), Computer Solution of Large Sparse Positive Definite Systems, Prentice Hall, Englewood Cliffs.

10. Gondzio, J. (1991), "A Method for Handling Dense Columns of Linear Programs in the Interior Point Algorithm", Presented at the International Symposium "Applied Mathematical Programming and Modelling", London.

11. Karmarkar, N.K. (1984), "A New Polinomial-Time Algorithm for Linear Programming", Combinatorica, Vol. 4, pp. 373-395.

12. Lustig, I.J., Marsten, R.E. and Shanno, D.F. (1989), "Computational Experience with a Primal-Dual Interior Point Method For Linear Programming", Technical Report SOR 89-17, School of Industrial Engineering and Operations Research, Georgia Institut of Technology, Atlanta.

INFORMATICA 4/91

PATTERN RECOGNITION USING KOHONEN MAPS AND MULTI-LAYER PERCEPTRONS

Tjaša Meško
Factuly of Electrical Engeneering and
Computer Science
University of Ljubljana

Keywords: multi-layer perceptrons,
initialization, a Kohonen map, clustering

Multi-layer perceptrons (MLPs) are now widely used for pattern recognition tasks such as speech recognition, handwritten character recognition, face recognition, etc. They were proven to generalize well to unseen data. Another kind of neural networks, namely, self-organizing feature maps have also been applied occasionally in pattern classification, but they were not that successful. In this study it is investigated how self-organizing feature maps could be useful in combination with MLPs as a tool for initializing the weights of a MLP. The purpose of the research was to reduce the amount of supervised training which is required to train MLPs.

PREPOZNAVANJE VZORCEV S KOHONENOVO MAPO IN Z VEČNIVOJSKIMI PERCEPTRONI. Večnivojski perceptroni se v zadnjem času pogosto uporabljajo pri prepoznavanju vzorcev (na primer govora), prepoznavanju pisav, prepoznavanju obrazov in podobno. Druga vrsta nevronskih mrež, Kohonenove mape, so bile tudi občasno uporabljene pri reševanju podobnih nalog, vendar rezultati so bili slabši. V tem članku je obravnavana možnost inicializacije uteži skritega nivoja trinivojskega perceptrona. Namen te raziskave je skrajšati čas učenja perceptrona.

1 Kohonen Self-Organizing Feature Maps

The self-organizing map belongs to the category of neural networks that use unsupervised training. This means that each time a new input is presented to the map, the desired output is unspecified. This type of neural networks is used to perform data compression, such as vector quantization and as it will be explained later, to reduce the amount of supervised training.

A vector quantizer is a mapping, q, that assigns to each input vector $\overline{x} = (x_1, x_2, \ldots, x_N)$, a codebook vector $\overline{c}_i = (c_{i1}, c_{i2}, \ldots, c_{iN})$

$$\overline{c}_i = q(\overline{x})$$

drawn from a finite set of codebook vectors

$$Q = \{\overline{c}_1, \overline{c}_2, \ldots, \overline{c}_M\}$$

where M is the number of codebook vectors. The quantizer q is completely described by the set of codebook vectors and it divides the input vector space into clusters

$$C_i = \{\overline{x} : q(\overline{x}) = \overline{c}_i\} \quad (1)$$

of input vectors which are mapped onto the i^{th} codebook vector. The distortion caused by reproducing an input vector $\overline{x}$ by a codebook vector $\overline{c}_i$ is given by

*A young researcher, employed in PAREX, d.o.o.

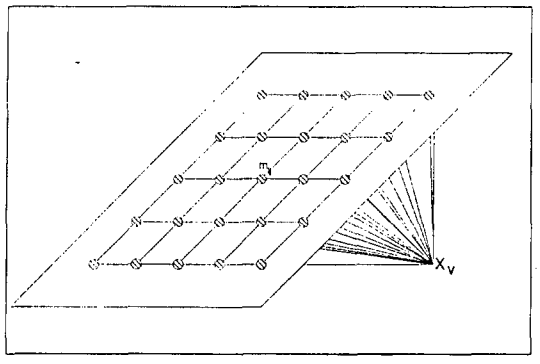


Figure 1: *Locations of map vectors in a square lattice. Vector $\overline{m}_{ij}$ can also be addressed as $\overline{c}_v$, where $v = (i-1)J + j$.*

$d(\overline{x}, \overline{c}_i)$, where d is assumed to be a Euclidean distance which is defined by equation 2.

$$d(\overline{x}, \overline{c}_i) = \sqrt{\sum_{n=1}^{N}(x_n - c_{in})^2} \qquad (2)$$

Kohonen's algorithm creates a vector quantizer by adjusting vectors which are typicaly arranged in a two-dimensional grid (usually a square or a hexagonal lattice). In this study square maps will be used (see figure 1) and the map vectors will be represented by their map coordinates (i, j). The vector at position (i, j) in the map will be addressed in two ways,

- $\overline{m}_{ij}$, $(i, j) = (1, 1), \ldots, (I, J)$
- $\overline{c}_v$, $v = 1, 2, \ldots, I \times J$

where $\overline{m}_{ij}$ equals $\overline{c}_{(i-1)J+j}$ and I, J are the map sizes.

The vector quantizer function $q(\overline{x})$ corresponding with a Kohonen map selects the codebook vector $\overline{c}_v$ which is closest to $\overline{x}$:

$$d(\overline{x}, \overline{c}_v) = \min_k d(\overline{x}, \overline{c}_k), k = 1, 2, \ldots, I \times J$$

In order to create the best codebook vectors, an iterative training is performed. During each iteration, a neighbourhood is defined around each vector of the map, as shown in figure 2. The neighbourhood NE(t) slowly decreases with time. At the beginning, the map vector components are initialized to small random values. Then, the following iterative procedure is applied whenever a new input vector $\overline{x}(t)$ is presented (t is the iteration number).

1. Compute the Euclidean distances $d(\overline{x}(t), \overline{c}_v(t))$, to all nodes $\overline{c}_v(t)$ according to the equation 2.
2. Select the node producing the minimum distance as the winning node $\overline{c}_v$.

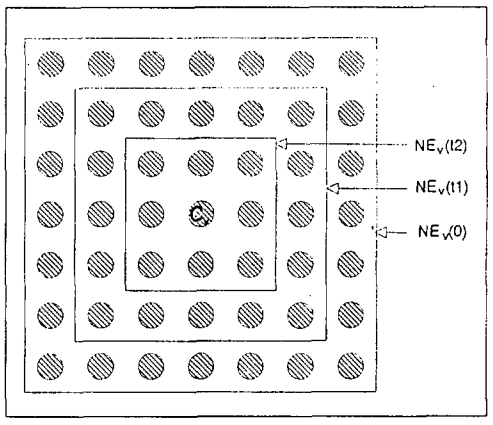


Figure 2: *Topological neighbourhood at different times as the feature map is formed. $NE_v(t)$ is the set of nodes considered to be in the neighbourhood of a node $\overline{c}_v$ at time t. The neighbourhood is decreased in size as time increases. In this example, $0 < t_1 < t_2$.*

3. Compute an updating factor $\alpha(t)$ and define a topological neighbourhood $NE_v(t)$.
4. Update the map vectors belonging to the topological neighbourhood of the winning node. For the adaptation of a map vector $\overline{c}_k$ the following formula is used,

$$c_{kn}(t+1) = c_{kn}(t) + \alpha(t)(x_n(t) - c_{kn}(t))$$

where $n = 1, 2, \ldots, N$.

The input vectors are taken from an input database, and are presented in a random order. The process is terminated as soon as the average distortion introduced by the vector quantization does not drop any more. Parameter $\alpha(t)$ is initialized to a value between zero and one, and is decreasing with time.

In this study an exponential rule for adapting α and for determining the topological neighbourhood [Brauer and Knagenhjelm, 1989] was used.

In order to use the Kohonen map for pattern classification, each map vector has to receive a label. The easiest way to achieve this is by applying the maximum a posteriory criterium: the labeled observations of a training set are presented to the map, and each node is labeled according to the number of observations of the different classes that were assigned to that node. Once the map is labeled, it can act as a pattern classifier.

The percentage of wrongly classified examples (the error rate) can be made as small as desired by introducing a large enough number of vectors. However, as it will be explained later, the training of a large map can be very time consuming and it is likely that such a map will not generalize properly to the unseen examples.

	6 × 6	9 × 9	12 × 12	15 × 15
train	74,55%	79,26%	81,02%	82,72%
test1	73,35%	77,59%	79,45%	80,79%
test2	75,47%	78,97%	79,85%	81,35%

Table 1: *Recognition rates on a BPC task, using different Kohonen maps.*

1.1 Results Using the Kohonen Map

The Kohonen map was tested as a pattern classifier on a Broad Phonetic Classification (BPC) task of Dutch spoken utterances. Here are the following five classes of the BPC task: *vowel*, *sonorant*, *fricative*, *burst* and *closure*. We have used a hand-labeled multi-speaker database of continuously spoken Dutch numerical strings, uttered by 30 different speakers (15 male and 15 female). The database consists of 300 different utterances (10 from each speaker). For training, 192 utterances from 24 speakers (8 from each speaker) are used and 108 utterances are used for testing. The test set is divided into 2 sets: *test1*, a multi-speaker test set (the remaining 48 utterances from the training speakers), and *test2*, a speaker independent test set (60 utterances from 6 new speakers). The speech signals were bandlimited to 4 kHz and sampled at a rate of 10 kHz. A 20-dimensional feature vector was extracted every 10 ms by means of an auditory model [Martens and Van Immerseel, 1990]. To take into account the dynamic nature of speech, several successive frames were presented simultaneously at the input of the map. In particular, we have used a feature vector consisting of three input frames: the first two frames were contextual frames located 40 and 20 ms ahead of the third frame which was the one to be classified. Therefore, each input vector consisted of 60 elements. The map vectors were labeled according to the maximum a posteriori criterium. The recognition rates for the different data sets are very similar, indicating an excellent generalization to unseen data. However, the feature-map cannot compete with a Multi-Layer Perceptron (MLP) trained by means of the back-propagation algorithm [Rumelhart et al., 1986]. Such a MLP obtains a recognition rate of 87,20% on the second test set [Depuydt et al., 1990]. The question was whether it would be possible to use the clusters obtained by the map to initialize the weights of a feed-forward net, and consequently to improve the supervised training time of that network.

2 Radial Basis Function Network

A traditional back-propagation network [Rumelhart et al., 1986] consists of nodes whose outputs are non-linear, differentiable squashing functions (typically sigmoid functions) f of the weighted sum of activations emerging from nodes on the previous layer. The output of node i is computed as follows,

$$o_i = f(\sum_j w_{ij} y_j + w_{i0})$$

where w_{ij} represents a connection weight, w_{i0} a bias variable, and y_j an output from a previous layer. The argument of f defines the following hyperplane in the input space:

$$\sum_j w_{ij} y_j + w_{i0} = 0$$

The hyperplanes defined by the different nodes constitute a set of class boundaries.

Searching for other ways of using the back-propagation algorithm, and thereby defining other kinds of class boundaries the idea of using *radial basis function* networks was introduced [Lowe, 1989]. The RBF network contains a hidden layer of m RBF units represented by the centres $\bar{c}_j$. The output layer consists of traditional summation units. Thus, the value of an output unit i is,

$$o_i(\bar{x}) = f(\sum_{j=1}^{m} \lambda_{ij} \Phi_j(\|\bar{x} - \bar{c}_j\|) + \lambda_{i0})$$

with f representing the sigmoid function, and Φ_j a RBF centered around a vector $\bar{c}_j$. Using the standard neural network terminology, the above formula for calculating the output of the i^{th} output-layer node in response to the p^{th} input pattern, o_{ip} can be stated as,

$$o_{ip} = f(\sum_{j=1}^{m} w_{ij} \Phi_j(\|\bar{x}_p - \bar{c}_j\|) + w_{i0})$$

In this study, a network consisting of a hidden layer of Gaussian nodes is considered, therefore, functions Φ_j are assumed to be,

$$\Phi_j(\|\bar{x}_p - \bar{c}_j\|) = \exp(-\sum_{n=1}^{N} \frac{(x_{pn} - c_{jn})^2}{2\sigma_{jn}^2})$$

where the parameters σ_{jn} represent the standard deviations of function Φ_j. The above network could be trained using the back-propagation algorithm with gradients:

$$\frac{\partial y_j}{\partial c_{jk}} = \frac{y_j(x_k - c_{jk})}{\sigma_{jk}^2}$$

$$\frac{\partial y_j}{\partial \sigma_{jk}} = \frac{y_j (x_k - c_{jk})^2}{\sigma_{jk}^3}$$

where $\Phi_j(\|\overline{x}_p - \overline{c}_j\|)$ is addressed as y_j. It was suggested that a network with RBF nodes could be trained by means of a layer-by-layer type of training (Gaussian nodes can be trained separately). In this case a computationally efficient way of determining the optimal weights to the output layer can be proposed [Renalds and Rohwer, 1989]. If y_{jp} is the output of RBF j given training input $\overline{x}_p$, then the output of output node i will be,

$$o_{ip} = f(\sum_j w_{ij} y_{jp} + w_{i0})$$

For optimal weights it holds that

$$E = 0.5 \sum_{ip} (o_{ip} - O_{ip})^2$$

is minimal, where O_{ip} is the desired value of the output node i. It can be shown [Renalds and Rower, 1989] that the optimal weights can be obtained as follows,

$$w_{ij}^* = \sum_k (\sum_p O_{ip} y_{kp}) M_{kj}^{-1} \qquad (3)$$

where M is the correlation matrix of the RBF outputs

$$M_{jl} = \sum_p y_{jp} y_{lp}$$

It is advised to use the pseudo-inverse of M to avoid possible singularities.

3 Initialization of Gaussian Nodes

With the error back-propagation algorithm, there is always a chance that the training gets trapped into a local minimum. The search of the optimal network configuration can also be very time consuming. Especially the determination of an optimal number of hidden units is a long process since the training has to be repeated for different amounts of hidden units. If there are too many hidden units, the generalization capabilities of the network might be reduced. If the number of hidden units is too low, the subspaces created by the network nodes cannot adequately model the class boundaries. Therefore, a dimensionality analysis of neural networks with one hidden layer of Gaussian hidden units and an output layer of conventional summation units was suggested [Weymaere and Martens, 1991]. The weights of the Gaussian nodes are initialized to values which are obtained by a modified k-means clustering algorithm [Wilpon and Rabiner, 1985] and the optimization procedure is performed in order to select the most effective set of Gaussian nodes. By performing a modified k-means clustering it is possible to obtain a fair description of the input data items by a limited number of clusters, each one being represented by a centre and a standard deviation vector. This parametric representation of the clusters can be used to initialize the hidden layer. The clustering can either be performed globally or per class. In the latter case [Weymaere and Martens, 1991], clusters are created for each class separately. During the k-means clustering the centre and standard deviation components are obtained for each class separately and for each number of clusters (1,..., predefined maximum). The cluster members tend to converge to that part of the input space that is covered by the examples of that class. Then the optimization procedure is carried out to select the optimal number of clusters for each class. The selected clusters are used to initialize the hidden layer in an almost optimal way.

We wondered whether it would also be possible to initialize the Gaussian nodes from the clusters obtained by a Kohonen map. A problem could be that the map was trained using all examples of the BPC task, and consequently that the map vector distribution was dominated by the dominating class (the vowels). In order to perform a clustering per class, we would have to construct five different maps (one for each class). Another problem is that by retaining only a few clusters from the map, only parts of the input space will remain well modelled. This is different in case of k-means clustering where the smaller cluster configurations (less centres) are determined to provide the best coarse representation of the data distribution over the entire input space. Due to the two problems stated above, there was doubt about the sensibility of using the Kohonen map cluster centres for the initialization of the Gaussian nodes. A property attributed to the Kohonen map is that it is not as much affected by noise and training inconsistencies as the k-means clustering is. Furthermore, the amount of nodes representing the different classes seems to be proportional to the number of examples of these classes in the training database. The map vectors tend to have the same distributions as the training database samples which is not the case for the clusters obtained by the k-means clustering algorithm.

The k-means clustering procedure (or the Kohonen map training) starts by selecting a small subset of the full training database (e.g. 50 times the number of clusters that are to be created). Clusters can be created globally or per class. In the former case an evaluation set is constructed which reflects the same a priori class probabilities as the full training database. In the latter case subsets of the different class sets are created. These sets (in case of global clustering there is

only one) are used to determine the centres and standard deviations of the candidate Gaussian units to be derived from the clusters.

The computation of the standard deviations assigned to the map vectors is performed as follows:

1. Compute exact cluster centres. In fact, the Kohonen map vectors divide the input space into clusters (see equation 1), but a map vector is not necessarily the centre of gravity of the cluster members.

2. Compute the standard deviations of the projections of the cluster members on the main axes.

3. Multiply all computed standard deviations by the same factor, which is determined in such a way that about 80% of the cluster members were located inside Gaussian output ellipsoid corresponding to a 0.5 output.

Once the cluster centres and the standard deviations are known, the optimization procedure [Weymaere and Martens, 1991] can be carried out. The optimization process is performed in n_p parallel paths. The parameter n_p is fixed in the beginning of the optimization procedure.

3.1 Results on the BPC Task

In order to evaluate the results obtained by initializing the network's hidden layer starting from a Kohonen map, the recognition rates of the obtained networks were compared to those of the corresponding networks which were initialized starting from the k-means clustering algorithm. Two comparisons were made: global clustering and clustering per class. (The results of the latter clustering method, using k-means clustering are reported in [Weymaere and Martens, 1991]). All the experiments were performed with n_p fixed to 3.

- The global k-means clustering and the 6×6 Kohonen map training were performed on a database consisting of 1800 samples. The 1800 samples reflected the same class probabilities as the full training database. The recognition rates of the initialized networks (as a function of the number of hidden nodes) are depicted on figure 3. The Kohonen map training took 77 minutes of CPU-time while k-means clustering took 75 minutes.

- In case of a clustering per class, 9 clusters per class (a 3×3 map in case of Kohonen clustering) were computed. In both cases a set of 500 examples of the same class was used for creating the clusters of that class. The recognition rates of the initialized networks, as a function of the number of hidden units, are shown in figure 4. The training of the Kohonen maps now took about 9 minutes, while the k-means clustering took no more than 2:20 minutes of CPU-time.

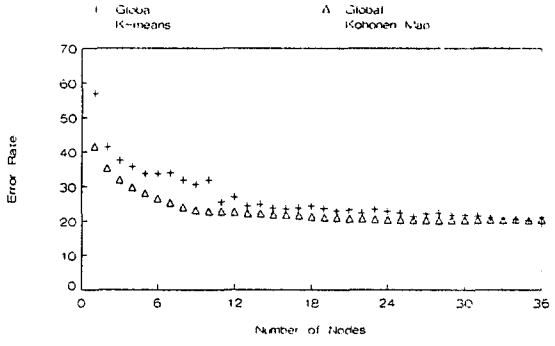


Figure 3: *Comparison of the recognition rates of networks using 36 global clusters (created by k-means algorithm/Kohonen map) to initialize Gaussian nodes.*

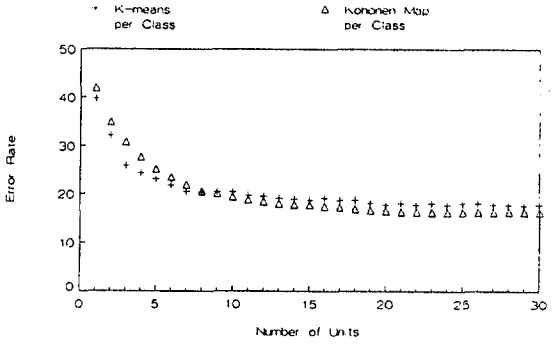


Figure 4: *Comparison of the recognition rates using 9 clusters per class for k-means clustering and for Kohonen map in order to initialize the Gaussian nodes.*

The Kohonen maps of sizes 12×12, 9×9 and 7×7 (which were already trained see subsection 1.1) were used to create clusters for the initialization of the Gaussian nodes. The parameter n_p was fixed to three and the maximum number of nodes was 60. In table 2 the recognition rates obtained by using three Kohonen maps for 30 and 60 hidden units are presented. The general conclusions are:

1. Using k-means clustering or Kohonen map clustering, it is possible to initialize MLPs with a Gaussian hidden layer to a near optimum point in

	144 clusters	81 clusters	49 clusters
30 nodes	83,8%	84,1%	83,0%
60 nodes	85,4%	84,4%	-

Table 2: *Recognition rates on a BPC task for different numbers of clusters which initialize Gaussian nodes.*

the weight space. The performance of the initialized network can be as large as 85,4% which is not far from the optimum performance of 89,20%, obtained with traditional 3-layer MLP trained with EBP.

2. The results obtained by clustering per class are substantially better than those obtained by global clustering (higher recognition rates and considerably less CPU-time).

3. The initial network performances obtained with k-means clustering and Kohonen map training are essentially the same, be it that k-means clustering is computationally more efficient. This superior computational efficiency is mainly devoted to the fact that all inter-sample distances required for k-means clustering can be computed (and stored) in advance. However, this advantage is lost as soon as the cluster datasets become larger.

4 Further training of the net

Once the Gaussian nodes are initialized, further training of the net can be carried out using the gradient descent method. The problem is that it is difficult to select proper learning rates and proper smoothing factors (for the adaptation of the output nodes' weights and for the adaptation of the Gaussian nodes' weights). If the choice of these parameters is inadequate, training can easily lead to a sub-optimum. Very few tests were run until now and the obtained results could still be improved.

The network with Gaussian units whose weights were obtained by our optimization procedure was tested on the three databases (training database, test set1, test set2). The weights to the output units were obtained according to equation 3. The optimization procedure used bpc_so as the training set. Networks with 20 and 30 Gaussian nodes obtained from a Kohonen map of 9×9 were used in this test. The network whose hidden layer consisted of 20 Gaussian nodes was trained for 8 cycles with a batch size of 1500 (this means that weights are adapted after 1500 input examples are presented). The network whose hidden layer consisted of 30 Gaussian nodes was trained for 13 cycles with the same batch size and for 20 additional cycles with a batch size of 3600 and a smaller learning rate. The results are presented in tables 3 and 4.

5 Conclusion

Experiments were carried out to investigate the capabilities of self-organizing feature maps (Kohonen maps) in a speech pattern recognition task. The conclusions of these experiments is that a labeled Kohonen map cannot compete with a feed-forward MLP trained on the same amount of labeled training examples.

	before training	after training
train	82,48%	88,70%
test1	80,58%	86,71%
test2	82,17%	86,94%

Table 3: *Recognition rates on a BPC task, using the network with 30 Gaussian nodes.*

	before training	after training
train	81,58%	87,27%
test1	80,21%	85,28%
test2	81,80%	85,50%

Table 4: *Recognition rates on a BPC task, using the network with 20 Gaussian nodes.*

Afterwards, it was investigated how Kohonen maps could be used to initialize the weights of a 3-layer MLP with a hidden layer of Gaussian units. It was found that initialization to a near optimum point in the weight space is feasible, especially if one starts from a set of small Kohonen maps each derived from examples of a particular output class. It was verified that Kohonen map clustering is a sensible alternative to k-means clustering, a technique which was introduced by Weymaere and Martens (1991) for the initialization of Gaussian networks.

The EBP-training of the initialized Gaussian networks lead to essentially the same recognition rates as were obtained with standard MLPs [Depuydt et al., 1990]. However, the initialization algorithm yields three major advantages:

1. The danger of getting trapped into a local minimum is reduced.

2. The required dimension (size) of the network can be determined without the need for EBP-training.

3. The training of the initialized network takes much less CPU-time than the traditional EBP-training of randomly initialized MLPs.

6 Acknowledgement

This study was made in Electronics Laboratory, University of Ghent in a periode from 1.2.1991 until 1.7.1991. I would like to thank my mentor Prof. Dr. Ir. Pipan from the University of Ljubljana, Faculty of Electrical Engineering and Computer Science for financial support. I am grateful to Prof. Dr. Ir. Vanwormhoudt for letting me work in his laboratory and

enabling me to use all its facilities. Special thanks are due to Dr. Ir. J.P. Martens for putting me on the right track, for correcting this report and for giving many useful advices. Thanks are also due to Ir. N. Weymaere for advising me on the literature, allowing me to use his programs and explaining me matters that I was not yet familiar with.

References

[1] Brauer,P. and Knagenhjelm,P. (1989). "Infrastructure in Kohonen Maps", Proc. IEEE ICASSP, Glasgow, May 1989, Vol. 1, pp. 647-650.

[2] Depuydt,L., Martens,J.P., Van Immerseel, L. and Weymaere, N. (1990). "Improved Broad Phonetic Classification and Segmentation with a Neural Network and a New Auditory Model", ICSLP, Vol. 2, pp. 1041-1044.

[3] Kohonen,T. (1989). "Self-Organisation and Associative Memory", Heiderberg, Springer Verlag, 3^{rd} Edition.

[4] Kohonen,T. (1990). "The Self-organizing Map", Proc. IEEE, vol. 78, no. 9, Sept. 1990, pp. 1464-1480.

[5] Lowe,D. (1989). "Adaptive Radial Basis Function Nonlinearities and the Problem of Generalization", IEE Conference on Artificial Neural Networks, 16-18 October.

[6] Martens,J.P. and Van Immerseel,L. (1990). "An Auditory Model Based on the Analysis of Envelope Patterns", ICASSP90, 402-406.

[7] Powell,M.J.D. (1985): "Radial Basis Functions for Multy-variable interpolation: a review", IMA Conference on algorithms for the approximation of functions and data, RMCS, Shrivenham.

[8] Renalds,S. and Rohwer,R. (1989). "Phoneme Classification Experiments Using Radial Basis Functions", IJCNN Washington.

[9] Rumelhart,D., Hinton,G. and Williams,R. (1986). "Parallel distributed processing: exploration in the microstructure of cognition", Vol. 1, MIT Press.

[10] Weymaere,N. and Martens,J.P. (1991). "A Fast and Robust Algorithm for Feed-forward Neural Networks", To appear in Neural Networks.

[11] Wilpon,J. and Rabiner, L. (1985). "A Modified K-means Clustering Algorithm for Use in Isolated Word Recognition", IEEE Transactions on Acustics, Speech and Signal Processing, Vol. 33, pp. 587-594.

FORMAL INFORMATIONAL PRINCIPLES

INFORMATICA 4/91

Keywords: circularity, decomposition, composition, construction of formulas, intelligent information, metaphysics, parallelization, particularization, principles of formula construction, sequentiality, spontaneity, universalization

Anton P. Železnikar
Volaričeva ulica 8
61111 Ljubljana

This essay shows the formalization possibilities of principles by which informational formulas, that is, informational entities occurring in the happening, eventful circumstances, are constructed in a spontaneous and circular way. In this sense, the formation (forming) of informational formulas becomes phenomenal by itself, not only by the spontaneous and circular use of the discussed principles, but also through the principles-own phenomenality. Formal informational system is an informationally arising system and, in this respect, it differs substantially from the concepts of the so-called well-defined, symbol-static, mathematically axiomatized systems. It could be said that an informational systems is spontaneously and circularly adapting to the situation and attitude (intention) of itself and its environment by the impact of itself and environmental information.

In this essay the following principles and their formalization are discussed and illuminated in a critically formative and subsequent way: spontaneity, decomposition (analysis), composition (synthesis), circularity, particularization, universalization, sequentiality (serialness), parallelization, structuring, organization, algorithmic information, straightforward information, informing, counter-informing, embedding, excluding, metaphysics, and intelligent information of formulas and formal systems. These principles are used within the informing of formulas and formal systems themselves, that is, as principles of their informational arising.

Formalni informacijski principi. Ta spis prikazuje formalizacijske možnosti principov, s katerimi se konstruirajo spontano in cirkularno informacijske entitete, tako kot se pojavljajo kot dogodja oziroma dogodkovne okoliščine. V tej smeri postane formacija (oblikovalnost) informacijskih formul tudi sama fenomenalna ne le s spontano in cirkularno uporabo obravnavanih principov, temveč tudi zaradi principom lastne fenomenalnosti. Formalni informacijski sistem je informacijsko nastajajoč sistem in v tej svoji značilnosti se bistveno razlikuje od t.i. dobro definiranih, simbolno statičnih, matematično aksiomatiziranih sistemov. Lahko rečemo, da se informacijski sistem spontano in cirkularno prilagaja situaciji in atitudi (intenci) samega sebe in svojega okolja z vplivom samega sebe in okoliške informacije.

V tem spisu se v kritično formativni in zaporedni obliki obravnavajo tile principi in njihova formalizacija: spontanost, dekompozicija (analiza), kompozicija (sinteza), cirkularnost, partikularizacija, univerzalizacija, posledičnost (zaporednost), paralelizacija, strukturiranje, organizacija, algoritmična informacija, premočrtna informacija, informiranje, protiinformanje, vmeščanje, izključevanje, metafizika in inteligentna informacija formul in formalnih sistemov. Ti principi se uporabljajo v okviru informiranja formul in formalnih sistemov samih, to je kot principi njihovega informacijskega nastajanja.

INTRODUCTION

In the essay *Principles of Information* [POI], several general rules concerning informational phenomenality (phenomenology) are treated, for instance, the spontaneity, circularity, informing, counter-informing, embedding, sequentiality (serialness), parallelism, structuring, organizing, and intelligence of information. In the present approach, our attempt will be to deliver as strict as possible formalization of the discussed principles for the informational formula development within a concise, self-sufficient theory. Already within the algebraic informational theory [IIA], the basic informational principles came to the surface: the arising of formulas was accompanied, for instance, by the so-called operator particularization and universalization; further, some (unconscious) principles of spontaneous and circular decomposition of operands and composition of formulas were applied. However, these principles were not treated in a systematic, conscious, and straightforward manner. The aim of this essay is to present formal principles for the development of an informational theory. The principles of this sort can also constitute the so-called axiomatic basis of a theory for the informational formula development.

The basic approach at the formula development will be a spontaneous and circular procedure, which starts by an initial informational marker, that is, a single operand symbol or by a set of markers. The formula development will be a subject of the so-called decomposition and composition principles which embrace some other principles, for instance, those belonging to particularization and universalization, serialization and parallelization, circularity and straight-forwardness, spontaneity and algorithmic approach, embedding and excluding of information, metaphysics and intelligence, etc. Hitherto, no attempt for a systematic and meaning treatise of this sort of spontaneous and circular decomposition and composition of informational formulas within a theory was made. Thus, we are standing in the front of the task to develop adequate principles in a formal, that is, axiomatic, theoretic, and symbolically appropriating way, preparing and designing the way fo a senseful theoretical and technological approach. As always in those situations, we are confronted with the problem, how to begin this systematic way, how to preserve the reasonableness, adequateness, and openness of formula development, that is, of their spontaneous and circular arising.

THE PRINCIPLE OF SPONTANEITY

Spontaneity of information is the first principle in the set of basic informational principles. Spontaneity of a formula development concerns the formula decomposition as well as the formula composition. A formula is nothing else than a model, scenario, depiction, or description of a real, mental, or artificial process, phenomenon, situation, or attitude, expressed in the form of informational operands, operators, parentheses, and punctation marks, for instance. It means that in the process of decomposition of a formula as an arising entity its inner components can be identified (brought to the surface or clarity) in a spontaneous way. In the process of composition of a formula as an arising entity its outer counterparts, that is, it concerning entities can be introduced or identified in a spontaneous way.

The basic question within the phenomenon of informational arising is, who or what is the actor of decomposing and composing spontaneity. Formally, the spontaneous actor of the development of formulas can be, for instance, a theory itself, a natural or artificial system, or a living actor demonstrating the faculty of spontaneous developmental action in such or another way. For instance, to a beginning, marking entity, the spontaneity can chose other marking entities which appear within or outside of the beginning entity and connect them operationally into a more complex formal system. In this system, the beginning entity can become a part or component of the decomposed and/or composed system. We argue that spontaneity of this sort is a regular (legal, permissive) principle of decomposition and composition of informational formulas.

Let α be an initial marker, that is, a yet undeveloped formula, i.e., operand marker. The

spontaneous development of the simple formula α can take the following spontaneous informational form:

(1) α;
$\alpha \models \alpha$;
$\alpha \models \xi, \eta, \ldots, \zeta$;
$\beta, \gamma, \ldots, \varepsilon \models \alpha$

In this system of formulas, α as the first formula represents the initial position. The second formula, $\alpha \models \alpha$, shows the beginning of the so-called metaphysical decomposition [IT2] of entity α. The third formula, $\alpha \models \xi, \eta, \ldots, \zeta$, shows the beginning of a composition in which α impacts entities $\xi, \eta, \ldots, \zeta$ in an informational way. In the last formula of system (1), $\beta, \gamma, \ldots, \varepsilon \models \alpha$, entity α is informationally impacted by entities $\beta, \gamma, \ldots, \varepsilon$. The point of system (1) is that formulas in the second, third, and fourth line came into existence in a spontaneous way. This spontaneity can be ascribed to a theory or generative system marked by $\mathfrak{T}$. Thus, formula system (1) can be clarified in the form

(2) α;
$\mathfrak{T} \models (\alpha \models \alpha)$;
$\mathfrak{T} \models (\alpha \models \xi, \eta, \ldots, \zeta)$;
$\mathfrak{T} \models (\beta, \gamma, \ldots, \varepsilon \models \alpha)$

where the acting entity $\mathfrak{T}$, by which formulas in the second, third, an fourth line of system (1) are introduced in a spontaneous way, is explicated.

The following principles will show how spontaneity can come to its action, how it comes to the surface in cases of applications of other informational principles.

THE PRINCIPLE OF DECOMPOSITION

The decomposition concerns always an unrevealed (non-disclosed) entity α which is, simultaneously, the most simple, basic formula α, representing a potentially complex, compound, or composed entity. In formula α, entity α acts (behaves, informs) as a unit which indicates yet unidentified or at some other place identified (determined, revealed, composed) entity (for instance, an expression of meaning, contents, understanding). In this way, a single α as expression is a sign, indicator (in German, das Anzeichen, in French, indice) which can be circularly decomposed, as it will be shown. The basic presumption is that α, in the process of its decomposition, is merely an initial formula of the form $\alpha \models \alpha$, that is,

(3) $\alpha \Rightarrow (\alpha \models \alpha)$

where $\Rightarrow$ is the operator of informational implication and $\models$ is the most general operator of informing (of α). We call $\alpha \models \alpha$ the metaphysics of α, indicating the α-inner informational process; thus, this sort of decomposition is a metaphysical one and must remain in the framework of $\alpha \models \alpha$. If β is a part of α, that is, $\beta \subset \alpha$, then $\alpha \models \alpha$ can be decomposed into

(4) $(\alpha \models \beta) \models \alpha$

where the circularity (cyclicity) of α, that is, $\dot{\alpha} \models \alpha$, is preserved. The metaphysical cycle $\alpha \models \alpha$ can be decomposed (de-constructed, revealed, analyzed, differentiated) to an arbitrary extent, depth, intention and also in the form of distinct cycles, for instance,

(5) $(\alpha \models \beta) \models \alpha$;
$((\alpha \models \beta) \models \gamma) \models \alpha$;
$(\alpha \models \beta, \gamma) \models \alpha$

etc., ad infinitum.

The basic rule of decomposition is the following: entity $\alpha \models \alpha$ can be decomposed serially (sequentially) and in a parallel way preserving the basic form $\alpha \models \alpha$ in any recursively decomposed form $(\alpha \models \beta) \models \ldots \alpha$. This kind of decomposition can occur also in a parallel form, can have parallel pathways (formulas in parallel) of the form $\alpha \models \ldots \alpha$.

If formula $(\alpha \models \beta) \models \alpha$ was one of the general decomposition principles, the other decomposi-

tion principle is

(6) $\alpha \models (\beta \models \alpha)$

which preserves the cyclic nature in regard to α. This duality leads to the presumption

(7) $(\beta \subset \alpha) \Rightarrow (((\alpha \models \beta) \models \alpha; (\alpha \models (\beta \models \alpha)))$

Further, one can have

(8) $((\gamma \subset \beta); (\beta \subset \alpha)) \Rightarrow$
$((((\alpha \models \beta) \models \gamma) \models \alpha); (\alpha \models ((\beta \models \gamma) \models \alpha)))$

etc., in a recursive and mixed recursive way.

Decomposition of $\alpha \models \alpha$ in one or another way means the spreading of formula $\alpha \models \alpha$ in its serial and parallel components and the mutual connection of components and the topic entity α. Through decomposition, formula $\alpha \models \alpha$ can blow up in an analytical and synthetical way, within itself as a serial and parallel circular system of formulas of different kinds. To these components, also outer components, not belonging to α, can be considered, for they can influence α, that is, its structure and organization. For instance, if β does not belong to (is not a part of) α, that is, $\beta \not\subset \alpha$, entity β can influence the metaphysics of α in the form

(9) $\beta \models (\alpha \models \alpha)$

Formula (9) is the typical position, where entity β is observed by metaphysics $\alpha \models \alpha$, that is by α. We see how informational phenomena can impact the arising of the initial formula $\alpha \models \alpha$. By decomposition (spreading, de-construction, delivering, distribution, dissemination), new informational entities can be introduced into the context of the metaphysical formation $\alpha \models \alpha$. The process of metaphysical decomposition is spontaneous and remains in the realm of entity α. The complexity of decomposition (spreading) of $\alpha \models \alpha$ depends solely on the degree and depth of distinguishing and differentiation of new components within α and their connectedness. As mentioned before, decomposition can be the subject of a theory $\mathfrak{T}$, of an outward observer ω, or of a technical tool τ (which possess an adequate technical understanding). In this way, $\mathfrak{T} \models (\alpha \models \alpha)$ or $\omega \models (\alpha \models \alpha)$ or $\tau \models (\alpha \models \alpha)$ are senseful formulas which explicate the action of decomposition of $\alpha \models \alpha$ by $\mathfrak{T}$, ω, or τ, respectively. We recognize how $\alpha \models \alpha$ as a theoretic, observational, or technological object is in no way only a tautological affair; it is the essential starting point at its actual and always potentially possible development.

THE PRINCIPLE OF COMPOSITION

The composition of formulas concerns different informational operands (entities) and their interweavement, that is, their operational connectedness. A composition of operand and operator components in the form of a system of formulas constitutes a new entity. While decomposition (spreading) is in fact merely a more detailed identification (clarification) of an already existing system, composition is the emerging of a new system, defined as informational composition of already composed entities. In fact, the composition and decomposition of formulas can act in a mutually senseful, spontaneous and circular way.

For instance, the metaphysical form $\alpha \models \alpha$ of entity α is an open self-informing (impacting, observing) system which can be decomposed (particularized) in an arbitrary (intentional, comprehensional, unpredictable, phenomenal) detail. If α informs openly, that is, $\alpha \models$, and if β is informed openly, that is, $\models \beta$, then β can become the observer of α. In this situation, one can set $\alpha \models \beta$. Now, formula $\alpha \models \beta$ can be decomposed in greater detail considering the components of α, β and $\alpha \models \beta$, and connecting them within the system $\alpha \models \beta$. We recognize, how decomposition and composition of components become interweaved procedures, proceeding from a hiddenly composed entity into greater detail by decomposition and composing a system in which detailed parts of a problem are coming to the formal surface.

If α and β are separated entities, that is, $\alpha \not\subset \beta$ and $\beta \not\subset \alpha$, then their composition can take several (spontaneous) basic forms. For instance,

(10) $(\alpha, \beta) \Rightarrow (\alpha \models \beta)$;
$(\alpha, \beta) \Rightarrow (\beta \models \alpha)$;
$(\alpha, \beta) \Rightarrow (\alpha \models \beta; \beta \models \alpha)$

etc.

The principles of decomposition and composition of formulas can be grasped also in a pure grammatical sense, for instance, in the form of a system including the following rules:

(**0°**) Symbol α (e.g., α, β, ... , ω) marks an operand; symbol $\models$ marks the most general type of operator; symbols '(' and ')' are parentheses.

(**1°**) α, $\alpha \models$, $\models \alpha$, and $\alpha \models \alpha$ are formulas.

(**2°**) Each formula can be used as an operand (recursiveness of formulas).

By rules (0°), (1°), and (2°) any formula can be decomposed and/or composed, starting by the symbol α. Let us have the following developmental steps, for example:

(11) α;
$\alpha \models$;
$\alpha \models \beta$;
$(\alpha \models \beta)$;
$(\alpha \models \beta) (\models$;
$(\alpha \models \beta) (\models \gamma$;
$((\alpha \models \beta) (\models \gamma)$;
$\models ((\alpha \models \beta) (\models \gamma)$;
$\gamma \models ((\alpha \models \beta) (\models \gamma)$

etc. A parallel case could be the following:

(12) α; β;
$\alpha \models$; $\models \beta$;
$\alpha \models \beta$; $\gamma \models \beta$;
$(\alpha \models \beta)$; $(\gamma \models \beta)$;
$(\alpha \models \beta) \models (\gamma \models \beta)$; $\delta \models (\gamma \models \beta)$

etc. The parallel has the meaning of introduction of parallel (newly appearing) formulas within a formula system.

THE PRINCIPLE OF CIRCULARITY

Circular schemes (scenarios, models, phenomena, processes, situations, attitudes) in the form of formulas can appear in various ways. By definition, an informational entity α itself is a circular phenomenon. The basic form of this circular phenomenality we call the metaphysics of α and denote it symbolically by $\alpha \models \alpha$. In a formula, the circularity of an entity α is expressed by the repeatedly appearing operand α in this formula. Formula $\alpha \models \alpha$, which marks the α's metaphysics, can be decomposed only by the extension of operator $\models$ in such a way that at the beginning and at the end of the decomposed (extended) formula there is operand α, for instance, $(\alpha \models \beta) \models \alpha$ or $\alpha \models (\beta \models \alpha)$. In the first case, process $\alpha \models \beta$ informs α; in the second case, entity α informs process $\beta \models \alpha$; in both cases entity α is involved circularly, but differently.

A different form of circularity is the parallel one. Parallel circular schemes are, for instance,

(13) $\alpha \models \beta; \beta \models \alpha$;
(14) $\alpha \models \beta; \beta \models \gamma; \gamma \models \alpha$

etc. Formula (13) shows the parallel circularity of the first degree, formula (14) of the second degree, etc. The so-called metaphysical circularity is the serial one, and is, accordingly to the extent of decomposition, of a higher degree (of greatest possible detail). Circularity of an operand α means its recursive appearance in formulas in a serial or parallel way.

The serial and parallel type of circularity occurs at the design of formulas in a natural (spontaneous) way. For instance, the metaphysical circularity of α, that is, $\alpha \models \alpha$, can be expressed (expanded, broadened) by its informing $\Im$, counter-informing $\mathbb{C}$, and embedding $\mathfrak{E}$ of information α, counter-information γ, and embedding information ε, respectively. Various kinds of schemes representing this metaphysical phenomenality of α can be appropriated. The most common (universal) one could be, for instance,

(15) $(((((\alpha \models \mathfrak{S}) \models \gamma) \models \mathfrak{S}) \models \varepsilon) \models \mathfrak{S}) \models \alpha$

However, several other metaphysical schemes can emerge during the process of decomposition, the serial as well as parallel ones.

Another, not necessarily metaphysical type of circularity of α concerns the so-called understanding $\mathfrak{U}$, which can appear as an intelligent component of informing $\mathfrak{S}$ within α. Usually, understanding $\mathfrak{U}$ produces (informs) a meaning μ_α of the understood situation inside of α and outside of α by α. Thus, a basic scheme of the understanding formula is

(16) $((\alpha \models \mathfrak{U}) \models \mu_\alpha(\xi)) \models \alpha)$

where ξ can concern α as well as an outside entity β or both of them. Of course, parallel metaphysical as well as understanding schemes of formulas can be constructed accordingly to the needs, concepts, and scenarios of the imagined reality. The process of the circular decomposition and composition of formulas is always a recursive one for some existing and arising operands and, in any case, parallel (new, additional) formulas for describing circular and non-circular (straightforward) phenomena can be introduced.

THE PRINCIPLE OF PARTICULARIZATION

Particularization of formulas is a procedure which proceeds from a general concept into more concrete detail of given phenomena in a decomposing way. Thus, in general, universal formulas are extended into corresponding particular forms. Particularization can be performed simply on the operator level when a general operator in a formula is replaced by an adequate particularized operator. Particularization concerns decomposition as well as composition of formulas (their sequentiality and parallelization) into more accurate detail in the sense of a descent to a sufficiently adequate concreteness. The terminal situation of a particularization is achieved through a stepwise procedure, descending in the imagined detail of a problem. Particularization is always a top-down construction of a formula (system), that is, from the beginning-general to the particular.

Particularization can also add formulas, operands, and operators to existing systems of formulas with the aim to concretize, analyze, explain, and complete the expression of concepts, scenarios, positions, attitudes, etc. In such procedure of formula development, other principles of informational development can be used (spontaneity, circularity, sequentiality, parallelism, etc.). The most simple act of particularization is the replacement of a general operator by the particularized one which, in an adequate manner, explicates the nature of informational impact between the concerned operands. In general,

(17) $(\alpha \models \beta) \Rightarrow (\alpha \models_{\text{part}} \beta)$

is a scheme of operator particularization. For instance, a theory, several concerned entities by themselves, or an outer observer can be imagined to be the actors of a particularization process.

THE PRINCIPLE OF UNIVERSALIZATION

While particularization concerns a top-down strategy of formula fitting concerning the problem, universalization is a bottom-up search from already concretized situations to the generalized ones. Universalization does not mean a simple replacement of a concrete formula by the general one; it is an introduction of new formulas which express general relations between operands and can be, afterwards, particularized again. Universalization introduces universal processes of informing into the context of a formula development. It performs as a bottom-up construction of formula system being on the way from the concrete to more general.

Particular cases can always signalize universal possibilities of their phenomenality. On the operator level, the universalizing principle can be the implication

(18) $(\alpha \models_{part} \beta) \Rightarrow (\alpha \models \beta)$

In this situation, the particular case $\alpha \models_{part} \beta$ is in no way neglected; it does not vanish necessarily from the observed system of formulas. But, the introduced general formula $\alpha \models \beta$ can now be decomposed in a new, different way in the form of a parallel phenomenon. Universalization can be understood as a spontaneous principle in the domain of a theory $\mathfrak{T}$, an observer ω, or a technical tool τ. In the universalization formulas, the general operator of implication $\Rightarrow$ can be particularized, for instance,

(19) $(\alpha \models_{part} \beta) \Rightarrow_{\mathfrak{T}} (\alpha \models \beta)$;
$(\alpha \models_{part} \beta) \Rightarrow_{\omega} (\alpha \models \beta)$;
$(\alpha \models_{part} \beta) \Rightarrow_{\tau} (\alpha \models \beta)$

yielding general schemes of the so-called operator universalization.

THE PRINCIPLE OF SEQUENTIALITY (SERIALIZATION)

Sequentiality concerns a serial development (composition, decomposition, circularity, particularization, universalization, etc.) of formulas under consideration. Circular sequentiality leads to the recursiveness of the occurring of operands. For instance, metaphysics, memory, self-constructive informational systems, preservation of a situation (a thing, body) or attitude (mind, consciousness), etc. can be recognized as sequential and, simultaneously, circular phenomena. The sequentiality of a phenomenon can be expressed also in a parallel circular way. Thus sequentiality in regard to the form of a formula or a formula system is twofold: serial (expansion of a formula by insertion of operands and operators) and parallel (expansion of a formula systems by formulas which within the left sides of operators $\models$ have operands already occurring in the existing formulas).

The development of a serial sequentiality in the formula $\alpha \models \beta$ can be shown in a stepwise manner by the following example:

(20) $\alpha \models \beta$;
$(\alpha \models \beta) \models \gamma$;
$(\alpha \models (\beta \models \delta)) \models \gamma$;
$(\varepsilon \models (\alpha \models (\beta \models \delta))) \models \gamma$

etc. A twofold circular expansion of the last formula in system (20) would be the case

(21) $((\varepsilon \models ((\alpha \models (\beta \models \delta)) \models \varepsilon)) \models \gamma) \models \alpha$

where the sequential circular expansion concerns α as well as ε which recursively appear in formula (21). While entity ε seems to form a proper subcycle of formula (21), entity α is a cycle which improperly enters into the ε-cycle, performing an interweavement of α-cycle and ε-cycle.

Sequentiality can concern the sequential depth structure of a formula, descending not only into a greater detail of a formula through serial analysis and structuring, but also interweaving sequential structures among themselves in any imagined way. At such structuring of a formula also some parallel cases of formulas can emerge, being the consequence of the depth analysis of an original formula. New, additional formulas can expose the faculty of the so-called parallel sequentiality which, in the most primitive case, can have the following form:

(22) $\alpha \models \beta; \beta \models \gamma; \gamma \models \delta$

etc. Parallel sequentiality can become circular, if to formula system (22), formula

(23) $\delta \models \alpha$

or any similar formula concerning the occurring operands in (22) is added.

The principle of sequentiality belongs to the most primitive approaches of the serial formula development, which considers the understanding of a formula and positions and attitudes of its operands. In short, this procedure of development can be called also the formula serialization.

THE PRINCIPLE OF PARALLELIZATION

Parallelization means introduction of parallel formulas by decomposition and composition of a formula; at this occasion parallel formulas emerge as a consequence of the depth analysis, synthesis, and splitting of a formula situation. The emerged formulas broaden a formula system and connect the arisen operand entities in a parallel or some other way. Through parallelization, a formula system becomes not only more complex, but also additionally interweaved, where interweavement concerns the arisen and the existing operand entities.

The principle of parallelization embraces the parallel composition and decomposition in a spontaneous, circular, particular, universal, and other ways. Parallelization means the growing in number and extent of informational components (operands and operators) and their interweavement through splitting the operand situations and attitudes. Parallel decomposition of a process, phenomenon, scenario, etc. is a splitting of the imagined (understood, comprehended, appearing) process, phenomenon, scenario entirety into parallel, interconnected entities. As an example, the following developmental steps within a parallel decomposition can occur:

(24) α;
$(\beta, \gamma \subset \alpha) \Rightarrow (\alpha \models \beta, \gamma)$;
$\alpha \models \alpha$; $(\alpha \models \beta, \gamma)$;
$(\delta, \varepsilon \not\subset \alpha)$; $(\delta, \varepsilon \models \alpha)$;
$\beta \models \delta$; $\gamma \models \varepsilon$

etc. The first formula α is the initial situation. By the second formula, entities β and γ within entity α are observed. This fact implies that components β and γ are informed by α. The third formula opens the possibility of metaphysical decomposition of α by $\alpha \models \alpha$ and explicates formula $(\alpha \models \beta, \gamma)$ which was implied by the second formula. The fourth formula observes that entities δ and ε are not the constituents of α, however, they impact α as the outer components. The fifth formula introduces the informing of δ by β and ε by γ, etc. A sequential (serial) consequence of the last parallel system can be the following:

(25) $((\delta, \varepsilon \models \alpha) \models \beta, \gamma) \models \delta, \varepsilon$

etc. If, consequently, entities δ and ε become parts (constituents) of α's metaphysics $\alpha \models \alpha$, system (25) closes into

(26) $(((\delta, \varepsilon \models \alpha) \models \beta, \gamma) \models \delta, \varepsilon) \models \alpha$

although in system (24), initially, $\delta, \varepsilon \not\subset \alpha$ was assumed.

Parallelization of formal systems can offer new interweavement of entities which can be in contradiction with the initial situation. A parallel decomposition brings to the surface new informational attitudes which may neglect or annul the initial conditions or force the observer to resolve the emerged contradictory situations, i.e., to change the observational conditions. Parallelization, as an introduction of parallelism of formulæ belongs to the most intelligible natural and artificial phenomena.

THE PRINCIPLE OF STRUCTURING

Structuring a system of formulas means to apply the principles of decomposition and composition, particularization and universalization, sequentiality and parallelization in a spontaneous and circular way. Structuring of a formula system can use, mix, and connect the discussed principles and those which follow. It can be said that the structuring as an active entity structures the already structured system to some extent, gains the arising of structure. Structuring is an informational component in the realm of information, where the principles of information [POI] are closed under information, that is, underlie the logic of informational arising.

Structuring means to restructure and to structure anew a structured informational system. Structuring acts like an informational component with characteristic informing, that is processing of structuring information which leads to a new structure. Structuring is a composed and complex use

of singular principles of information.

Structuring can get its concrete sense as a particular, that is, informationally determined form or process of shaping, arising, emerging of an informational system in a structural way. For instance, structuring $\mathfrak{S}$ as a structuring activity and potentiality belongs to the structure entity σ which informs $\mathfrak{S}$ how to structure a primitive or system entity α in question. Thus, primordially,

(27) $(\sigma \models \mathfrak{S}) \models \alpha; \alpha \models \sigma$

where σ observes α, that is, its structure σ_α and where σ_α serves as the reference for structuring α. Thus, one can imagine,

(28) $(\alpha \models \sigma) \models \sigma_\alpha;$
$(\sigma_\alpha \models \sigma) \models \mathfrak{S};$
$\mathfrak{S} \models \alpha$

or, in a cyclic form,

(29) $((((\alpha \models \sigma) \models \sigma_\alpha) \models \mathfrak{S}) \models \alpha) \models \sigma$

The last scheme represents two perplexed cycles concerning the entity to be structured, that is, α, as well as the structuring entity σ. This scenario performs as long as the structure entity σ is not satisfied with the structure σ_α. In this function, entity σ can use a reference or dynamic understanding of σ_α for the decision making how to structure α and since when to stop the structuring of α. A concrete process of structuring can be particularized to any necessary extent (detail), thus, specifying the process of structuring of the entity in question.

THE PRINCIPLE OF ORGANIZATION

A strict distinction between structure and organization remains vague. In some way, however, it is possible to differentiate the phenomena of structuring and, as a consequence of a structure, the arising of organizational relations, for instance, the interweavement of structural informational entities, by which the nature of connectedness, impact, dependence, conditionality, etc. comes into existence. Organization means a supplement in the understanding of structure of a formula system. Informationally, structuring causes the so-called organizational relations which are nothing else than the additional, to the structure occurring informational processes. Thus, the arising organization seems to be a consequence of the structure introduced by an informational entity.

What could the organization of a formula system, which is a structure of operands, operators and formulas, mean at all? How could the clear difference between the structure and organization of a formula system be observed?

The structure of a formula system could be grasped as a visible arrangement, dispósition, and appearance of operands, operators, and their formulas. Of course, the structure of a formula hides also the meaning of structuring and the comprehension of occurring formula components. The structure of a formula is a consequence of structuring (grammatical, syntactic) rules, which govern the composition of formula constituents (operands and operators).

The organization of a formula concerns its depth structure which is not only grammatical. Organization is related to the interweavement of meaning and understanding of occurring operands and operators, their parallel and circular connectedness and to the impact of occurring organizational forms with processes of decomposition and composition, that is, with the arising of informational formulas. In a semiotic way, organization is an arrangement of operands and operators in a semantic and pragmatic manner, is also a semantic and pragmatic (spontaneous) disposition of operands and operators in a formula. The observing organizational view can cause the arising of new formulas within a formula system. For instance, explaining a system of formulas through the addition of formulas represents an organizational decomposition of the system. Organizing seems to be a particular view of structuring and vice versa. Both, structuring and organizing, are compositional as well as decomposing principles which can enrich, broaden, advance, and complete

a system of formulas under investigation.

The first look at a formula system can be merely structural; afterwards, through study, analysis, and development of the system, the first look becomes more and more organizational. If structure is a sort of system identification, organization is the understanding of the structural meaning. This is the well-known cycle, which comes into existence between the structural meaning and organizational understanding. This cycle gains the development of structure and organization of a formula system in a structuring and organizing way.

Let us imagine the following structure (σ) and organization (o) development system ϑ for a formula system φ:

(30) $(\vartheta \models ((((\varphi \models \sigma) \models (\mathfrak{S} \models \varphi)) \models o) \models (\mathfrak{D} \models \varphi))) \models \vartheta$

Formula (30) describes a circular development system (ϑ) with two significant development subprocesses, which are structuring ($\mathfrak{S} \models \varphi$) and organizing ($\mathfrak{D} \models \varphi$) of the formula system φ. However, the development of a formula system φ can be expressed also in a traditional parallel form, for instance, as

(31) $\varphi \models \sigma(\varphi)$;
$\sigma(\varphi) \models \mathfrak{S}$; $\mathfrak{S} \models \varphi$;
$(\sigma(\varphi), \varphi \models o(\varphi)) \models \mathfrak{D}$; $\mathfrak{D} \models \varphi$;
$(\mathfrak{S}, \mathfrak{D} \models \vartheta)$; $\vartheta \models \varphi$

where $\sigma(\varphi)$ and $o(\varphi)$ are the intentional structure and organization of formula system φ and, $\mathfrak{S}$ and $\mathfrak{D}$ the structuring and organizing mode of the development system ϑ, respectively. This development system seems to be φ-circular and the improvement (learning, adaptation) of ϑ is governed by structuring $\mathfrak{S}$ and organizing $\mathfrak{D}$, however, still remaining within the cycle of development of φ.

THE PRINCIPLE OF ALGORITHMIC INFORMATION

Algorithmic information is a well-determined, self-sufficient entity which can be repeatedly applied (understood, used) as a clear, data-stable, definitive recipe (procedure, process) for solving a certain type of problem. Particular algorithmic information is, for instance: mathematical algorithms for various purposes (solving of equations, calculating values, deducing and proving of theorems, developing theories, searching for axioms and principles, etc.); computer programs expressed in different programming languages (well-structured application programs, expert systems, artificially intelligent tools, etc.); and theories of sciences, in general, which have to deliver reliable and repeatable results and predictions. The principle of algorithmic, that is, well-defined, disciplinarily structured, or scientifically doctrinaire information pervades the entire realm of mathematics, computer science, and sciences in general. Furthermore, a certain piece of algorithmic information guaranties that the process it describes can be effectively transferred to a technical tool (automatic equipment, robot, computer, etc.). The contents and structure of an algorithmic information can be always applied, understood, or learned by a materially realized technical tool or a mental system.

Algorithmic information belongs to the so-called realm of data. In contrast to information, data informs in an identical (repeatedly, definitively regular) way and depends only on data, that is, on well-defined arguments. Algorithmic information is not informational yet in the sense of informational arising. Algorithmic information performs as a functional or procedural determined data structure for handling data. But, data δ as a function or argument [IT2] is nothing else than

(32) $\delta \Leftrightarrow_{Df} (\delta \models, = \delta)$

which initial metaphysical form is $\delta = \delta$. Algorithmic information performs as a tool for solving a kind of problem irrespective of the values of arguments which inform (impact) algorithmic in-

formation (procedure, program) in differently occurring situations.

THE PRINCIPLE OF STRAIGHTFORWARD INFORMATION

The main characteristics of a straightforward information is that it is not explicitly circular. For instance, formula $\alpha \models \beta$ is straightforward while $\alpha \models \alpha$ is not. Irrespective of the extent of decomposition, a metaphysical information as a whole cannot be straightforward. Implicitly, the circularity of straightforward information occurs. For instance, within $\alpha \models \beta$, informational entity α as well as β are circular. Also, the parallel circularity of the system $\alpha \models \beta; \beta \models \alpha$ does not violate the principle of straightforwardness. Thus, consequently structured parallel systems can keep the principle of straightforwardness. This principle excludes any explicitly circular scheme (formula, scenario). To achieve the state of straightforwardness of a formula system, circular formulas can be decomposed in an adequately parallel way. However, this could mean to reduce a natural scenario into an artificial model.

In most cases, both principles of circularity and straightforwardness will be used at the decomposition and composition of formulas together with other principles. The straightforwardness means the hiding, concealment, and placing out of sight the circularity of operands. This happens in a natural way at speaking, writing, and thinking in any language. It may happen that clearly parallel cases of an implicitly circular structure are interrupted. The best examples of this sort are the so-called concepts of words in a dictionary. A word is rarely defined (conceptualized) by itself. However, in a semantic net of words, circularity (tautology in a transitive sense) always exists. Thus, the principle of the so-called straightforward information is, in fact, the vagueness (hiding) of the circular nature of informational entities in question.

The sciences are inclined to the straighforwardness because it enables the abstraction by simplification and reduction, or a satisfactory explanation of otherwise circularly (recursively) perplexed phenomena. The principle of straightforward information concerns a unidirectional, shortened, and decisively unambiguous way to the solution of a problem.

THE PRINCIPLE OF INFORMING

Informing of things is the most basic principle in the realm of information. It says that an informational entity α informs and is informed and thus implies the system

(33) $\alpha \models; \models \alpha$

This system determines entity α in its actual and potential entirety [IT2], irrespective of its material, mental or, in general, phenomenal nature.

Informing of entity α, that is, $\Im_\alpha$ or simply $\Im$, can be observed explicitly; it is implicitly present in operator $\models$, that is, in formula $\alpha \models$ as well as formula $\models \alpha$. Thus, entity α can be decomposed in regard to its informing $\Im$. The following straightforward implication seems to be reasonable:

(34) $\alpha \Rightarrow (\Im \subset \alpha; \alpha \models \Im; \Im \models \alpha)$

It says that informing $\Im$ is a constituent of α, however, α and $\Im$ inform each other and are informed by each other. Another form of informing $\Im$ within α can be conceptualized circularly, for instance, as

(35) $\alpha \Rightarrow ((\alpha \models \Im) \models \alpha)$

This circular scheme presumes that informing $\Im$ is a subordinated (subjected) component of α and, certainly, that $\Im$ informs α circularly. Thus, the following implication in regard to the initial situation described by (35) seems to be reasonable:

(36) $((\alpha \models \mathfrak{Y}) \models \alpha) \Rightarrow (\mathfrak{Y} \subset \alpha; \alpha \subset \mathfrak{Y};$
$((\mathfrak{Y} \models \alpha) \models \mathfrak{Y}));$
$((\mathfrak{Y} \models \alpha) \models \mathfrak{Y}) \subset ((\alpha \models \mathfrak{Y}) \models \alpha)$

Here, the cycle of informing $((\mathfrak{Y} \models \alpha) \models \mathfrak{Y})$ is subordinated to the main (origin) cycle $((\alpha \models \mathfrak{Y}) \models \alpha)$. Both, entity α and its informing $\mathfrak{Y}$ are parts of each other within the eventfulness (happening of informing) of α. That α and $\mathfrak{Y}$ are parts of each other means that they mutually and perplexedly exchange the roles of subject and object: if in one of the events of α, α impacts $\mathfrak{Y}$, then in another event, $\mathfrak{Y}$ impacts α, or they may impact each other even simultaneously, that is, in the way of a proper (simultaneous) interaction.

It is to say that informing $\mathfrak{Y}$ of α is not or cannot be exhausted in a decomposing way. Entity α and its informing $\mathfrak{Y}$ can hide various forms of entities and to them belonging informing. The most general components are, for instance, counter-informing of counter-information and embedding of information by the so-called embedding information. These cases will be treated separately in the next two sections and together with informing as the whole, in the framework of metaphysics of an informational entity.

THE PRINCIPLE OF COUNTER-INFORMING

The principle of counter-informing is an attempt to grasp the arising or coming of information into existence through an explicit (revealed, disclosed) happening of an informational event. Within informing $\mathfrak{Y}$ of entity α, counter-informing $\mathfrak{C}$ is the recognizable entity within $\mathfrak{Y}$, which is directly concerned with the arising of information, called counter-information γ. Counter-informing $\mathfrak{C}$ is nothing else than informing of counter-information γ, of course, within the entity α and its informing $\mathfrak{Y}$, respectively.

While it has just arisen or it is still arising, counter-information γ as a distinctive entity within α which is not informationally connected (embedded in respect) to entity α yet. It performs as an isolated entity within α as long as it is not embedded (informationally connected) by virtue of the so-called embedding information ε. The similar holds for counter-informing $\mathfrak{C}$ within informing $\mathfrak{Y}$. So far, counter-informing $\mathfrak{C}$ is an isolated part of informing $\mathfrak{Y}$ of entity α.

We see that to be a part of something, but to be not connected to something, means simply to be not embedded in something. This is the characteristics of an informationally isolated part of something. It means that an entity can produce a subentity within itself which does not impact the entity itself yet. The arisen, isolated part of an entity has to be connected to or embedded into the entity to contribute to the arising of the entity as an integrated, whole thing of its parts.

We have to introduce a particular operator for the so-called not connected informing, for instance, $\not\subset_{con}$. Thus, at the arising of counter-informing $\mathfrak{C}$ within informing $\mathfrak{Y}$ and at the arising of counter-information γ within entity α, we have the following situation:

(37) $\mathfrak{Y} \models \mathfrak{C}; \mathfrak{C} \subset \mathfrak{Y}; \mathfrak{C} \not\subset_{con} \mathfrak{Y};$
$\alpha \models \gamma; \gamma \subset \alpha; \gamma \not\subset_{con} \alpha;$
$(\mathfrak{Y} \models \alpha) \models \mathfrak{Y}$

The isolated counter-informing $\mathfrak{C}$ performs in regard to $\mathfrak{Y}$ as a strange (unconscious), to $\mathfrak{Y}$ yet unobserved entity, which produces the unobserved counter-information γ. It means that in the first step of counter-informational arising there is no observing in the form

(38) $\mathfrak{C} \models \mathfrak{Y}; \gamma \models \alpha$

or, explicitly,

(39) $\mathfrak{C} \not\models \mathfrak{Y}; \gamma \not\models \alpha$

Operator $\not\models$ explicates the non-informing or non-observing situation. The observing in the sense of formula (38) occurs after the so-called embedding of counter-informing $\mathfrak{C}$ into informing $\mathfrak{Y}$ and counter-information γ into entity α in question. After the process of embedding, both counter-informing $\mathfrak{Y}$ and counter-information γ lose the

status to be counter-informational entities. They become regular informing and regular information, respectively.

Counter-information must be understood as the possible increase of an informational entity when the process of embedding of counter-informational components into the original informational entity is taking place. Counter-information γ is that informational entity which arises out and within of the informational entity α by virtue of its open informing $\Im$, that is, as a parallel open system

$$(40)\quad \alpha \Rightarrow ((\alpha \models \Im) \models; \models (\alpha \models \Im); (\Im \models \alpha) \models; \models (\Im \models \alpha))$$

The adequate circular, parallel, and open system would be, for instance,

$$(41)\quad \alpha \Rightarrow (((\alpha \models \Im) \models \alpha) \models; \models ((\alpha \models \Im) \models \alpha); ((\Im \models \alpha) \models \Im) \models; \models ((\Im \models \alpha) \models \Im))$$

The counter-informing as the phenomenon of informational arising can take place within system (40) or (41).

THE PRINCIPLE OF EMBEDDING

The embedding of information can be understood as the act of reception, observation and/or, finally, the perception of information by an entity. The process of embedding is an accumulative and integrative process in informational sense regarding the entity in question. It is the basic principle of appropriation of information by information. By embedding, the outside or inside unconnected information comes into consideration by the entity which embeds.

The embedding itself proceeds from the basic informational assumption $\alpha \Rightarrow (\models \alpha)$. The question is: which information comes from others and the entity itself as that which has to be embedded? In other words, what are the other things and the thing itself for the thing itself? Or, how to the general question *What is a thing for other things?* the antisymmetric (inverse) question *What are things for the thing in question?* can be considered as the origin question concerning the embedding of information?

The concept of embedding proceeds from formula $\models \alpha$ in the sense

$$(42)\quad \alpha, \beta, \gamma, \ldots \models \alpha$$

This formula illustrates a (partial) process by which entities α, β, γ, ... are in the process of embedding, where entity α observes them and can be impacted by their informing. The potentiality of embedding can be expressed, for instance, by formula

$$(43)\quad \alpha, \beta, \gamma, \ldots \models (\models \alpha)$$

or by implication

$$(44)\quad (\alpha, \beta, \gamma, \ldots \models (\models \alpha)) \Rightarrow (\models (\alpha, \beta, \gamma, \ldots \models \alpha))$$

Embedding of information means the modus of information for itself (in contrary to information for others).

THE PRINCIPLE OF EXCLUDING

How can informational entity β be or how can it stay excluded in regard to an entity α? Entity β is excluded in regard to α if it does not inform α in any way. Thus the exclusion principle could be symbolized by

$$(45)\quad \beta \not\models \alpha$$

Operator $\not\models$ is a symbol for a certain non-informing. However, *how does β not inform α?* could be another question. In fact, an explicit form of non-informing between entities can be understood already as a particular case of informing between entities. Immediately we speak out a case of non-informing, we have to do with a particular form of informing.

The proper case of the exclusion of information could be the vanishing, disappearing, forgetting

and, certainly, non-observing of certain information.

THE PRINCIPLE OF METAPHYSICS

In regard to the entity as entity, the metaphysical as a thing has the meaning of to be entirely concerned with the entity itself, that is, with the thing within the thing itself, however, still in an open, environmentally impacted manner. For instance, an observing thing as observer always produces its metaphysical information, that is, it observes the observed thing by its metaphysics or metaphysical components. To understand the principle of metaphysics μ, one has to observe several metaphysical phenomena. If α is one of the metaphysical components within entity β, we use the symbolic notation $\alpha \subset_{\mu} \beta$. Then the following can be observed:

knowledge $\subset_{\mu}$ belief;
belief $\subset_{\mu}$ truth;
truth $\subset_{\mu}$ logic;
logic $\subset_{\mu}$ language;
language $\subset_{\mu}$ mind;
mind $\subset_{\mu}$ central_nervous_system

etc. We see how metaphysical components are (hierarchically) nested (already partly embedded) in each other. For instance, belief roots partly in knowledge (is informed by it); truth is a consequence of belief and knowledge. The conscious of truth impacts the arising of logic. Language is the eventfulness for knowledge as well as belief and truth. Mind is the home of language, where language (for instance, speech acts, writing) creates mind. The central nervous system arises under the impact of the mentioned metaphysical components. All of these components are circular and spontaneous, structured and organized, autonomous and interactive.

However, not only mental phenomena (knowledge, belief, truth, logic, language, mind, central nervous system, etc.) are understood to be metaphysical. Regardless of its nature, every thing has its metaphysics, that is, the inner phenomenality (processing, form) of thing [IT2]. From the point of view of language, the human logic is always metaphysical. The mind as the entire mental phenomenality is metaphysical too from the point of view of the central nervous system. For instance, the consciousness of man can never surpass the mind in a non-metaphysical way. Thus, for man, there is not possible to think outside of a his/her metaphysical (mind-concerning, neuronal, autopoietic) background. Metaphysics is the beginning and the end of each informational (philosophic, rational, irrational, scientific, etc.) phenomenality. It can develop, arise, reach any metaphysical point, concern, achievement, development as an open, environmentally impacted system, but cannot surpass its instant potentiality and actuality in the framework of its instantaneous openness and (informational) arising.

Metaphysics is one of the four specific modes of informing of things. It is senseful to list these modes to keep the insight into the problem of informing and the context of informing in which metaphysics can be distinguished among other phenomena concerning the informing of things in general. In the case of entity α, we clearly distinguish:

(46) $\alpha \models; \models \alpha$ as entity α *itself;*
$\alpha \models$ as entity α *for others;*
$\models \alpha$ as entity α *for itself;* and
$\alpha \models \alpha$ as entity α *in itself.*

As we see, the essential attributes of these distinctions are: *itself, for others, for itself,* and *in itself.* All four modes of informing are open:

(47) $((\alpha \models; \models \alpha) \models; \models (\alpha \models; \models \alpha))$;
$((\alpha \models) \models; \models (\alpha \models))$;
$((\models \alpha) \models; \models (\models \alpha))$;
$((\alpha \models \alpha) \models; \models (\alpha \models \alpha))$

Now, these modes of informing of entity α can be applied to metaphysics $\alpha \models \alpha$ as an informing entity. Thus, it is possible to distinguish the following:

(48) $((\alpha \models \alpha) \models; \models (\alpha \models \alpha))$ as *metaphysics itself;*
$(\alpha \models \alpha) \models$ as *metaphysics for others;*
$\models (\alpha \models \alpha)$ as *metaphysics for itself;* and
$((\alpha \models \alpha) \models (\alpha \models \alpha))$ as *metaphysics in itself.*

These formulas can have sense in the framework of the metaphysical decomposition.

Proceeding from $\alpha \models \alpha$ as a metaphysical situation of α means first of all that α informs α and that formula $\alpha \models \alpha$ is not only an ontological expression in the sense α *is* α, but also α *is not* α. Why the second, negative statement is senseful? Because $\alpha \models \alpha$ means that α changes itself through its informing and that a comparison of a state of α by its subsequent state never gives a statement of the type α *is equal* α, that is $\alpha = \alpha$. This comment explains the reasonableness of introducing the metaphysical, in fact, ontological formula $\alpha \models \alpha$.

The metaphysical is always concerned with the auto-cyclicity in an open way. The so-called metaphysical components, that is, components of a metaphysical entity α, have the metaphysical structure in itself. For instance,

(49) $(\beta \subset_{\mu} \alpha) \Rightarrow ((\beta \models \beta) \subset (\alpha \models \beta))$

or also

(50) $(\beta \subset_{\mu} \alpha) \Rightarrow ((\alpha \models (\beta \models \beta)) \models \alpha)$

If γ is an outer component which impacts α, then, in general, it can also impact the components of α. In this case, formula (50) becomes

(51) $(\gamma \models \alpha; \beta \subset_{\mu} \alpha) \Rightarrow ((\gamma, \alpha \models (\beta \models \beta)) \models \alpha)$

In the last formula, entity γ, which impacts α, is a distinguished outer component, which is not impacted by α. It means that α, together with its component β, as the metaphysical observer of γ, does not impact γ so far. In every case of observation, a thing α can observe its environment and/or the thing itself only in a metaphysical way, that is, through (the decomposition of) $\alpha \models \alpha$.

THE PRINCIPLE OF INTELLIGENT INFORMATION

Intelligent information, marked by ι, belongs to metaphysical information. It appears as as a part of a thing's metaphysics, that is, as the so-called intelligence of things (beings, minds, programs) which can understand things and can be understood by intelligent things. Thus, things appear to an intelligent information as to be intelligible, that is, apprehensible by the observing thing. Intelligent information performs in an observing and self-observing way. Concerning its arising, emerging, coming into existence, and unforeseeableness, it is thrown into counter-informational situations where it tries to solve some environmentally and by itself impacted problems, searching for an intelligent meaning and understanding of a solution. As any proper information, intelligent information is an unforeseeable phenomenon in the sense to delivering solutions, which can be recognized as intelligent by itself and other intelligent observers. At this point, intelligent information encounters the domain of the so-called communal intelligence, that is, by a certain community governed intelligent information.

Intelligent information possesses its own intelligent counter-informing $\mathfrak{S}_{\iota}$, which is the component for unforeseeable production of new information. When observed, intelligent information can be carefully decomposed (analyzed, specifically composed, particularized, etc.), proceeding from its initial metaphysical situation $\iota \models \iota$. Within the phenomenon of intelligent information several other components can be observed which contribute to its further identification (decomposition, de-construction).

What does intelligent information produce, how does it arise? Intelligent information arises in connection with the occurring, appearing unpredictable situations when it is thrown into the happening and eventfulness of its environment and itself. Besides classical metaphysical components which are intelligent informing $\mathfrak{I}_{\iota}$, intelligent counter-informing $\mathfrak{S}_{\iota}$ and intelligent embedding $\mathfrak{E}_{\iota}$, producing (informing) intelligent information ι, intelligent counter-information γ_{ι} and intel-

ligent embedding information ε_ι, respectively, several other forms of intelligent information can be considered. Let us try to show a classification of intelligent information on the global level, where understanding and the informed meaning are coming into the foreground. Intelligent information ι as metaphysical information is decomposed through its initial situation $\iota \models \iota$. Intelligent information has to include a sort of understanding $\mathfrak{U}$ (roughly, intelligence) which substantially impacts the arising of ι. Intelligent information is always an understanding information, that is,

(52) $\mathfrak{U} \subset \iota ; (\iota \models \mathfrak{U}) \models \iota$

The next question is, how does understanding $\mathfrak{U}$ impact intelligent information ι? What does $\mathfrak{U}$ produce? Understanding $\mathfrak{U}$ as the producer of intelligent information ι produces the so-called intelligent meaning μ_ι for intelligent information ι in a metaphysical way. That means

(53) $(\mathfrak{U} \models \mu_\iota) \models \mathfrak{U}; \mu_\iota \subset \iota$

Commonly, meaning μ_ι belongs to the category of informational (e.g. linguistic) concepts which as products of an understanding do not belong to understanding information, that is, $\mu_\iota \not\subset \iota$. However, in a general scenario of understanding $\mathfrak{U}$ within intelligent information ι, there is

(54) $((\iota \models \mathfrak{U}) \models \mu_\iota) \models \iota$

In a different situation, formula

(55) $(\iota \models (\mathfrak{U} \models \mu_\iota)) \models \iota$

or even formula

(56) $((\iota \models ((\mathfrak{U} \models \mu_\iota) \models \mathfrak{U})) \models \iota$

can be appropriate. The last formula shows the direct nesting of cycle $(\mathfrak{U} \models \mu_\iota) \models \mathfrak{U}$ within cycle $\iota \models \iota$. Several other combinations of cycles concerning intelligent entities ι, $\mathfrak{U}$, and μ_ι are possible.

Certainly, when arisen, meaning μ_ι performs as metaphysical information $\mu_\iota \models \mu_\iota$ impacting understanding $\mathfrak{U}$, yielding

(57) $(\mu_\iota \models \mathfrak{U}) \models \mu_\iota$

Thus, in a further way expanded form of formula (56) could be, for instance,

(58) $((\iota \models (((\mathfrak{U} \models \mu_\iota) \models \mathfrak{U}) \models ((\mu_\iota \models \mathfrak{U}) \models \mu_\iota)) \models \iota$

In this sense, intelligent information approaches to the attitude to be more and more cycled, where informational cycles can overlap in a parallel way too. Scenarios corresponding to concrete cases can be constructed by means of the demonstrated informational language.

CONCLUSION

Information as a phenomenon appears as nothing else than the connection (interaction, influence) between things performing as informational entities. Formalization of the connection (impact, phenomenality) from one thing to another thing is an informational phenomenon by itself and performs as regular information. Formulas and systems of formulas are informational entities of the observer who observes things and their interaction (impacts). In this sense, formulas and systems of formulas are understood to be the interface between the reality of things and observers (things) of things. Informational entity is a symbolism which appears between the observed and observing thing set (understood) by the observing thing.

It becomes evident that an intelligent informational phenomenon, as a consequence of performing, behavior and being of an intelligent thing, can be satisfactorily symbolized only by an intelligent informational formula. What is an intelligent formula and how does it behave? In short, intelligent formula performs as intelligent information. It means that it must be metaphysical in informational way, that it must be observing and self-observ-

ing, arising, emerging and coming into existence, in short, adaptable to the circumstances and to its throwness into unforeseeable situations and attitudes. In contrast to a mathematical formula or a computer program, informational formula is a changeable and emerging structure so far. Only in particular cases, it can take the role of a static, data, mathematical, or program structure to serve as a mechanical tool for an unchangeable and dedicated purpose. In contrast to a mathematical notion, informational formula has its own intention, skill, and rationality accordingly to which it can develop, emerge, arise autonomously through impacts coming from the world, itself, and the thing to which it mediates (communicates, informs).

The eventfulness of a formula or a formula system images the events belonging to things which a formula or a formula system depicts. In this function, formula is the symbolic coping of situation existing between the thing in question and its world. In parallel to the thing in the world, the formula is thrown in the world of observation and its own behavior (functionality, adequateness against a situation), impacting the observing thing, its metaphysics. Thus, a situation, its symbolical expression, and observing constitute a system of entities impacting them not only consequently but rather in an interactive manner. Symbolization of phenomena is in the position to become information for others, for observers of things and for observers of observers. Within occurring positions and attitudes of events higher informational forms and processes emerge, for instance, the phenomena of intention, consciousness, self-consciousness, unconsciousness, and other intelligently structured entities.

The construction of formulas through the use of the discussed principles opens the way to spontaneous and circular possibilities for the emerging of formulas. Principles themselves are—as we could see—openly arising informational entities which, in a concrete case, are involved into situations of decomposition, composition, particularization, universalization, informing, counter-informing, embedding, etc.

One of the aims of this essay is to reveal the importance of informational understanding of formulas, to give them the emerging faculty, by which things themselves and their interactions can be symbolized in a more natural, adaptable way. An informational formula is nothing else than an interface between the world and the thing in the world. It is an emerging symbolic depiction of being-in-the-world. It comes closer to the reality of a thing through its emerging structure than any statically structured symbolism (mathematical, programming language) could ever come. The principles of this essay have to be understood also as a changeable, emerging information for informational development of formulas which have the ability to depict real situations and real attitudes of things in question.

REFERENCES

[POI] Železnikar, A.P., *Principles of Information,* Cybernetica **31** (1988), 99-122.

[IIA] Železnikar, A.P., *An Introduction to Informational Algebra,* Informatica **14** (1990) *1,* 7-28.

[IT2] Železnikar, A.P., *Informing of Things II* (in Slovene), Informatica **15** (1991) *3,* 29-43.

A COMMENT

PLANIRANJE S PROLOGOM

INFORMATICA 4/91

Matjaž Likar in Nikola Guid
Tehniška fakulteta Maribor
Inštitut za računalništvo in informatiko
Smetanova 17, Maribor

Keywords: planning, search, heuristics

POVZETEK: Članek opisuje realizacijo sistemov za planiranje nedovisno od domene v programskem jeziku prolog. Za predstavitev problemov je uporabljena konceptualizacija v prostoru stanj. Planiranje poteka po treh različnih strategijah: planiranje z iskanjem v globino (sistem PLAN_S), hevristično planiranje z omejitvami (sistem PLAN_H) in planiranje z omejitvami (sistem PLAN_C). Rezultati eksperimentov kažejo na to, da je časovna zahtevnost sistema PLAN_S reda $O(b^d)$, kjer je b povprečni faktor vejitve in d dolžina plana. Sistema PLAN_H in PLAN_C uporabljata dodatene omejitve, ki izhajajo iz poznavanja posameznih domen. Rezultati eksperimentov nakazujejo, da je njuna časovna zahtevnost reda O(bd). Pri vseh treh sistemih je ugotovljena močna korelacija med časom planiranja in številom obravnavanih stanj. Za reševanje kompleksnejših problemov planiranja iz realnega sveta je potrebno uporabiti katero izmed boljših strategij.

ABSTRACT: Realization of the domain independent planning systems in PROLOG is described. State space conceptualization for problem representation is used. Three different planning strategies are implemented: planning with depth first search (system PLAN_S), heuristic planning with constraints (system PLAN_H) and planning with constraints (system PLAN_C). The results of the experimental measurements indicate that the time complexity of the system PLAN_S is $O(b^d)$, where b is an average branching factor and d plan length. System PLAN_H and PLAN_C use additional constraints coming out from better domain knowledge. Experimental measurements show that their time complexity is O(bd). Strong corelation between planning time and number of the visited states is established at all three systems. Better strategies should be used for solving planning problems with real life complexity.

1 UVOD

Začetki raziskav na področju planiranja v umetni inteligenci segajo v pozna petdeseta in zgodnja šestdeseta leta. Planiranje najprej pomeni način za poenostavitev zapletenih problemov, pri katerem iz rešitev enostavnih problemov razvijemo rešitve kompleksnih problemov. V tem času (1959) nastane eden izmed prvih sistemov za planiranja GPS ("general problem solver") [CHAP87,ALLE90]. V začetku sedemdesetih let se pojavijo nove definicije planiranja [BARR83,GEOR90], ki se ujemajo z običajnim razumevanjem te besede [RAND83,VERB87], in smatrajo plan kot natančen program akcij, planiranja pa kot razmišljanje o akcijah pred njihovim izvajanjem [NILS90]. Nadaljnji razvoj umetne inteligence prinese nove logične formalizme [GEOR90] (predikatni račun, situacijski račun, modalna logika), ki se izkažejo uporabni tudi na področju planiranja. Najbolj poznane sisteme za planiranje skupaj s približnimi časi njihovih nastankov prikazuje tabela 1 [TATE90]. Bistvene lastnosti, ki jih srečamo pri teh sistemih, so:

1. sposobnost izpeljevanja novih zaključkov iz že poznanih dejstev o problemu ter relacij in omejitev, ki veljajo za določeno domeno,
2. nelinearnost, s katero se izognemo iskanju vseh možnih vrstnih redov akcij znotraj plana,

3. hierarhičnost, ki proces planiranja razdeli na več nivojev, glede na stopnjo razgradnje problema,
4. sposobnost obravnavanja spremenljivk,
5. vključevanje dodatnih omejitev, s katerimi omejimo iskanje,
6. sposobnost ponovnega iskanja planov zaradi zunanjih sprememb, ki so nastale neodvisno od subjekta, ki izvršuje plan,
7. neodvisnost od domene.

Tabela 1 Kronologija nastankov poznanih sistemov planiranja

obdobje	ime sistema
1960-1965	GPS, QA3
1965-1970	REF-ARF, STRIPS
1970-1975	ABSTRIPS, HACKER, PLANEX, MACROPS, WARPLAN, HEARSAY, INTERPLAN
1975-1980	NOAH, NONLIN, MOLGEN, NONLIN+, NASL,
1980-1985	OPM, DEVISER, SIPE, PLANX 10, TWEAK
1985-1990	OPLAN, PRS, SCRAPS, CHEF, PLEX, GTD, FORBIN, PRIAR, PRODIGY/EBL

V tabeli 2 je podano, katere izmed zgoraj naštetih lastnosti so vključene v nekatere poznane sisteme za planiranje [WILK88].

Tabela 2 Lastnosti sistemov za planiranje

	lastnosti						
sistem	1	2	3	4	5	6	7
STRIPS							+
HACKER							+
ABSRIPST			+				+
NOAH		+	+	+			+
NONLIN		+	+	+			+
DEVISER		+	+	+	+		+
MOLGEN		+	+	+	+		
SIPE	+	+	+	+	+	+	+

Prvi sistemi za planiranje so predvsem eksperimentalne narave in imajo za cilj formalizacijo človekovih sposobnosti razmišljanja o prihodnjih akcijah. V sistemih STRIPS in NOAH je ideja planiranja realizirana kot iskanje potrebnih akcij in njihovo razvrščanje znotraj plana. Pri sistemu STRIPS, ki so ga razvijali ob koncu šestdesetih in v začetku sedemdesetih let, poteka to iskanje samo na enem nivoju, kar se pri kompleksnejših problemih izkaže kot premalo učinkovit pristop [WILK88]. Ta slabost je odpravljena v sistemu ABSTRIPS [SACE74], kjer poteka proces planiranja na večih nivojih **hierarhije abstraktnih** prostorov ("hiearchy of abstraction spaces"). Prvi klasični sistem za planiranje NOAH (1975) je že sposoben poiskati nelinearne in hierarhične plane, pri čemer uspešno obravnava tudi spremenljivke. Njegov naslednik NONLIN (1977) obvlada **vračanje** ("backtracking") [CHAP87,WILK88]. Začetki osemdesetih let prinesejo mnogi nove sisteme za planiranje: MOLGEN (1981) [STEF81a,STEF81b], DEVISER (1982) [CHAP87], SIPE (1983) [WILK88] in TWEAK (1984) [CHAP87]. Bistvo teh sistemov je, da upoštevajo dodatne **omejitve** ali **zahteve** ("constraints") nad spremenljivkami, s katerimi operiramo v procesu planiranja, omogočajo planiranje za doseg **konjunktivnih ciljev** ("conjunctive goals") in **ponovno planiranje** ("replanning") ob neuspešnem izvrševanju plana.

V tem članku želimo opisati realizacijo sistemov za planiranje in na podlagi eksperimentalno dobljenih podatkov ugotoviti, kakšna je časovna zahtevnost ter kateri faktorji vplivajo na učinkovitost planiranja. Poleg tega obravnavamo tudi lastne izkušnje in rešitve, ki so specifične za planiranje v prologu.

2 ZAHTEVNOST PLANIRANJA

Kompleksnost prostora stanj, ki ustreza problemu planiranja, je večinoma eksponentna funkcija velikosti problema [KORF87]. To potrjuje tudi kratka analiza velikosti prostora stanj za klasični problem planiranja v **svetu blokov** ("blocks world").

Število stanj, tj. različnih razporeditev n blokov v k skladov ($k \in \{1, 2, ..., n\}$) je enako vrednosti nepredznačenega Lahovega števila $L'_{n,k}$ (pri danem n in k) [AIGN79]:

$$L'_{n,k} = \frac{n!}{k!}\binom{n-1}{k-1} \qquad (1)$$

Celotno število različnih stanj za n blokov, tj. s(n), je enako vsoti vseh nepredznačenih Lahovih števil pri danem n (enačba 2, tabela 3):

$$s(n) = \sum_{k=1}^{n} L'_{n,k} \qquad (2)$$

Zgornja meja števila različnih stanj za svet blokov pa je:

$$s(n) = O\left((3/2)^{(n-1)}\, n!\right), \qquad (3)$$

iz česar vidimo, da gre za eksponentno rast prostora stanj. S povečevanjem števila blokov pa ne narašča le število različnih stanj, temveč tudi število vseh različnih akcij a(n). Če za premikanje blokov uporabljamo akcije: *unstack*, *stack* in *move* [GENE87], potem je:

$$a(n) = n \sum_{k=1}^{n-1} k\,(k+1)\, L'_{n-1,k} \qquad (4)$$

Povprečno število akcij, ki jih lahko izvedemo v nekem stanju, oz. **faktor vejitve** b(n) ("branching factor"), dobimo kot kvocient

števila vseh možnih akcij a(n) in števila vseh možnih stanj s(n) (tabela 3).

Tabela 3 Kompleksnost prostora stanj za svet blokov

n	s(n)	a(n)	b(n)
1	1	0	0.000
2	3	4	1.333
3	13	30	2.308
4	73	240	3.288
5	501	2 140	4.271
6	$4.051\ 10^{3}$	$2.130\ 10^{4}$	5.258
7	$3.763\ 10^{4}$	$2.351\ 10^{5}$	6.246
8	$3.944\ 10^{5}$	$2.854\ 10^{6}$	7.237
9	$4.597\ 10^{6}$	$3.782\ 10^{7}$	8.228
10	$5.894\ 10^{7}$	$5.434\ 10^{8}$	9.220
20	$3.277\ 10^{20}$	$6.283\ 10^{21}$	19.172
30	$1.980\ 10^{35}$	$5.771\ 10^{36}$	29.147
50	$3.772\ 10^{68}$	$1.853\ 10^{70}$	49.119
100	$2.422\ 10^{164}$	$2.400\ 10^{166}$	99.089

Pri večjem številu blokov se faktor vejitve približa vrednosti n, zato velja:

$$b(n) = O(n) \qquad (5)$$

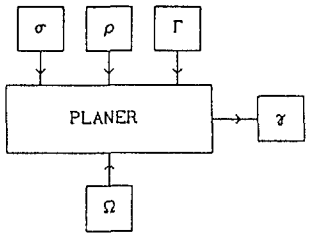
Zahtevnost problema planiranja in s tem tudi čas planiranja pa ni odvisen samo od števila blokov, oz. faktorja vejitve, marveč tudi od dolžine iskanega plana d, ki je lahko od 0 do 2(n-1). Funkcijski izraz, ki povezuje faktor vejitve in dolžino plana s časom planiranja, je odvisen od strategije planiranja.

3 REALIZACIJA SISTEMOV ZA PLANIRANJE

Študijo sistemov za planiranje smo izvedli v treh korakih. Najprej smo poiskali ustrezen model sistema, ga zatem implementirali na računalniku in preizkusili pri reševanju različno zahtevnih problemov.

3.1 Model sistema

Model sistema za planiranje (slika 1) smo zgradili na osnovi sheme iz [GENE87]. V sistem PLAN_S smo vgradili lastnosti 1, 4 in 7 (glej tabelo 2), kar pomeni, da gre za linearni sistem planiranja neodvisnega od domene z možnostjo obravnavanja spremenljivk. Sistem je v omejenem obsegu tudi sposoben izpeljevanja novih dejstev iz predhodno poznanih dejstev. Sistema PLAN_H in PLAN_C vsebujeta dodatne kontrolne mehanizme za povečevanje učinkovitosti (lastnost 5).

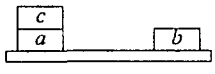


Slika 1 Zgradba sistema za planiranje

Oznake, ki nastopajo na sliki 1, pomenijo:

- σ - **označevalnik začetnega stanja** ("initial state designator"),
- ρ - **označevalnik cilja** ("goal designator") ali **ciljna relacija** ("goal relation"),
- Γ - množica **označevalnikov akcij** ("action designators") (enostavne akcije in zaporedja akcij),
- Ω - podatkovna baza s stavki o začetnem stanju, cilju in uporabnih akcijah,
- γ - plan (zaporedje akcij), s katerim ob izvršitvi v začetnem stanju dosežemo ciljno stanje.

Začetno stanje predstavlja posnetek opazovanega sveta v trenutku, preden začnemo izvajati akcije iz plana (slika 2).

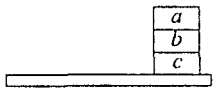


Slika 2 Primer začetnega stanja

Začetno stanje opišemo s seznamom dejstev, ki veljajo v tem stanju:

initial([clear(c),on(c,a),table(a),clear(b),table(b)]).

Cilj ("goal") je lahko katerokoli dosegljivo željeno stanje opazovanega sveta [GENE87] (slika 3). To je tisto stanje, ki ga v realnem svetu vzpostavimo tedaj, ko opravimo vse v planu zahtevane akcije. Definiciji cilja lahko ustreza več stanj.



Slika 3 Primer ciljnega stanja

Cilj, da je npr. v svetu treh blokov blok *a* na bloku *b* in hkrati blok *b* na bloku *c*, zapišemo kot:

goal([clear(a),on(a,b),on(b,c),table(c)]).

Za opisovanje akcij smo uporabili zapis, v katerem je podan seznam **pogojev** ("preconditions") za izvedbo akcije, seznam *pozitivnih* **učinkov** ("positive effects"), oz. dejstev, ki postanejo resnična po izvršitvi akcije, in seznam **negativnih učinkov** ("negative effects"), oz. dejstev, ki po izvršitvi akcije ne veljejo več.

action *action_name(argument1,argument2,...)* :
preconditions ==> *positive effects* # *negative effects.*

S tem formatom so implicitno rešeni **problemi ozadja** ("frame problems"), prav tako pa je v njem zajeta tudi strategija **poravnave stanj** ("state alignment") [GENE87,WINS84]. Tako smo se izognili spremenljivkam za označevanje stanj, ki so npr. potrebne pri planiranju z uporabo resolucije. Definicije akcij niso del sistema za planiranje, ampak predstavljajo njegove vhodne podatke (Γ na sliki 1). Podati jih moramo za vsako domeno posebej. Akcijo razlaganja (*unstack*) iz blokovnega sveta smo v naših sistemih definirali kot:

action u(X,Y) : [on(X,Y),clear(X)] ==>
[table(X),clear(Y)] # [on(X,Y)].

Sistem za planiranje iz vhodnih podatkov: označevalnika začetnega stanja σ, ciljne relacije ρ, množice označevalnikov akcij Γ in podatkovne baza Ω poišče plan γ, ki predstavlja rešitev problema planiranja tedaj in le tedaj, če zadovoljuje pogoja [GENE87]:

1. γ mora biti element množice označevalnikov akcij Γ:
$$\gamma \in \Gamma.$$
2. Podatkovna baze Ω mora logično implicirati, da izvršitev (Do) plana γ v začetnem stanju σ ustvari stanje, ki ustreza relaciji ρ:
$$\Omega \models \rho(Do(\gamma,\sigma)).$$

3.2 Implementacija sistemov

Sistem PLAN_S

Sistem PLAN_S smo realizirali na principu algoritma iskanja v globino v smeri naprej od začetnega stanja proti ciljnemu stanju (slika 4) [BRAT86,STER86]. Jedro tega sistema je rekurzivna procedura **plan(+Initial,+Goal,+History,+Depth,-Plan)** (s predpono + so označeni vhodni argumenti, z - pa izhodni argument). Pred klicem procedure moramo poznati začetno stanje, ciljno stanje in omejitev iskanja v globino. Argument **History** vsebuje seznam že poznanih stanj v delno zgrajenem planu in je pri prvem klicu prazen ([]). Iskanje v globino omejuje argument **Depth**. Robni pogoj rekurzije je izpolnjen tedaj, ko se začetno in ciljno stanje ujemata (**match**). Kot rezultat dobimo **Plan** - seznam akcij. Znotraj procedure **plan** kličemo proceduro za spremembo stanja **do_forw**, Z **do_forw(-Action,+OldState,-NewState)** preverimo, ali se pogoji **Pre** za izvedbo akcije **Action** ujemajo s starim stanjem **OldState**.

```
plan(S1,S2,_,_,[]) :- match(S1,S2).

plan(S1,S2,History,OldDepth,[Action|Plan]) :-
    OldDepth > 0,
    do_forw(Action,S1,NewS1),
    not(known(NewS1,History)),
    NewDepth is OldDepth - 1,
    plan(NewS1,S2,[S1|History],NewDepth,Plan).

do_forw(Action,OldState,NewState) :-
    action Action : Pre ==> Add # Del,
    match(Pre,OldState),
    del_all(Del,OldState,Temp),
    add_all(Add,Temp,NewState).
```

Slika 4 Planiranje z iskanjem v globino v smeri naprej

Zatem tvorimo novo stanje **NewState**, tako da iz starega stanja zbrišemo negativne učinke **Del** in mu dodamo pozitivne učinke **Add** akcije **Action**.

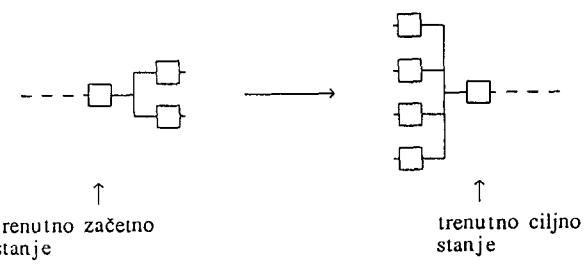
Sistem PLAN_H

Iskanje plana na slepo (s poskušanjem) in **vračanje** ("backtracking") v situacijah, kjer ugotovimo, da nadaljnje planiranje ni več mogoče, se je pri kompleksnejših primerih izkazalo kot premalo učinkovito. Sistem PLAN_H (slika 5) predstavlja nadgradnjo predhodnega sistema.

```
plan(_,S1,S2,[]) :- match(S1,S2).

plan(forward,S1,S2,[Action|Plan]) :-
    do_forw(Action,S1,NewS1,S2),
    direction(NewS1,S2,Direction),
    plan(Direction,NewS1,S2,Plan).

plan(backward,S1,S2,[Action|Plan]) :-
    do_back(Action,S2,NewS2,S1),
    direction(S1,NewS2,Direction)
    plan(Direction,S1,NewS2,P),
    concatenate(P,[Action],Plan).


do_forw(Action,OldState,NewState,Goal) :-
    choose(Action,OldState,NewState,Goal).

do_back(Inverse,OldGoal,NewGoal,Initial) :-
    choose(Action,OldGoal,NewGoal,Initial),
    inverse(Action,Inverse).

choose(Action,S1,NewS1,S2) :-
    findall(Act,(action Act : Pre ==> Add # Del,
                match(Pre,S1),legal(Act,S1,S2)),Actions),
    find_best(Actions,Action,S1,NewS1,S2).
```

Slika 5 Dvosmerno hevristično planiranje

S pomočjo procedure **direction(+S1,+S2,-direction)** smo realizirali dinamično izbiranje smeri planiranja. Na vsakem koraku iskanja plana ocenimo, ali terja manj naporov iskanje v smeri naprej ali v smeri nazaj. Če je faktor vejitve v trenutnem začetnem stanju manjši kakor faktor vejitve v trenutnem ciljnem stanju, potem naredimo en korak v smeri naprej (**do_forw**) (slika 6), v nasprotnem primeru pa v smeri nazaj (**do_back**).



Slika 6 Primer izbire smeri planiranja naprej

Sistemu PLAN_H smo dodali omejitev, s katero izmed vseh akcij, ki so izvedljive v nekem stanju, izberemo le dovoljene (**legal**) akcije. Akcija je v sistemu PLAN_H dovoljena v nekem stanju samo tedaj, ko njena izvršitev ne onemogoči nadaljnjega planiranja brez vračanja. Zaradi omejitve **legal** so učinki akcij **odvisni od okoliščin** ("context-dependent effects"), v katerih akcijo izvršujemo. Proceduri **legal** in **direction** je potrebno sestaviti za vsako domeno posebej. Pravimo, da gre za **kontrolne strategije iskanja za specifične domene** ("domain-specific search control strategies") [WILK88]. Iskanje plana v sistemu PLAN_H poteka po strategiji **vzpenjanje** ("hill-climbing"). Izmed vseh dovoljenih akcij v naslednjem koraku s hevristično ocenjevalno funkcijo izberemo najboljšo akcijo (**find_best** na sliki 5). Od hevristične ocenitvene funkcije zahtevamo, da nikoli ne **preceni** ("overestimates") dejanskih stroškov preostalega (še nedograjenega) plana. Izpolnitev te zahteve zagotavlja, da sistem PLAN_H vedno najde optimalno rešitev.

Sistem PLAN_C

Pri hevurističnem planiranju v sistemu PLAN_H je iskanje vseh potencialno možnih in glede na okoliščine dovoljenih akcij prostorsko in časovno zahteven proces, kajti učinkovita (hitra) izračunljivost hevristične ocenjevalne funkcije je slabo združljiva z njenim učinkovitim delovanjem. Sistem PLAN_C vsebuje samo omejitev **legal** in se pri izbiri naslednje akcije zadovolji s prvo najdeno dovoljeno akcijo (slika 7). S tem smo bistveno zmanjšali čas planiranja, odrekli pa smo se temu, da na vsakem koraku poiščemo najboljšo akcijo. Plani, ki jih najde sistem PLAN_C, zato niso optimalni.

```
choose(Action,S1,NewS1,S2) :-
    action Action : Pre ==> Add # Del,
    match(Pre,S1),
    legal(Action,S1,S2),
    del_all(Del,S1,Temp,add_all(Add,Temp,NewS1).
```

Slika 7 Izbira prve dovoljene akcije

V tabeli 4 je podan pregled omenjenih realiziranih sistemov s kratkim opisom strategije planiranja.

Tabela 4 Kratek opis realiziranih sistemov

sistem	opis sistema
PLAN_S	planiranje z iskanjem v globino naprej
PLAN_H	dvosmerno hevristično planir. z omejitvami
PLAN_C	dvosmerno planiranje z omejitvami

3.3 Eksperimenti

Vse eksperimente smo izvedli na računalniku PC s procesorjem 80386/80387 (25 MHz) in s pomnilnikom RAM velikosti 4 MB. Od sistemov smo zahtevali, da najdejo pravilen plan, poleg tega pa smo njihovo kvaliteto ocenjevali še po naslednjih treh kriterijih:

- čas planiranja,
- število obravnavanih stanj in
- dolžina najdenega plana.

Te podatke nam izpišejo sistemi v trenutku, ko je planiranje zaključeno. Za vse tri sisteme PLAN_S, PLAN_H in PLAN_C smo pripravili enak vzorec z 20 primeri iz sveta blokov, ki se razlikujejo po številu blokov ($n \in [1, 2, ..., 5]$) in dolžini plana d. V vzorcu so bili enakomerno zastopani primeri, ki so lažje rešljivi s planiranjem v smeri naprej, kot tudi primeri, ki so lažje rešljivi v smeri nazaj. Pazili smo tudi na to, da se v rešitvah - planih približno enako pogosto pojavljajo akcije *unstack*, *stack* in *move*. Pri sistemu PLAN_S smo za vsak problem planiranja najprej izbrali omejitev globine, ki je enaka dolžini optimalnega plana, zatem pa še omejitev 2(n-1), s katero lahko rešimo najbolj neugoden primer za n blokov. Sistema PLAN_H in PLAN_C smo dodatno preizkusili še na primerih z večjim številom blokov ($n \in [10, 20, 30, 50, 100]$), ker smo želeli ugotoviti, kje so meje njunih zmogljivosti.

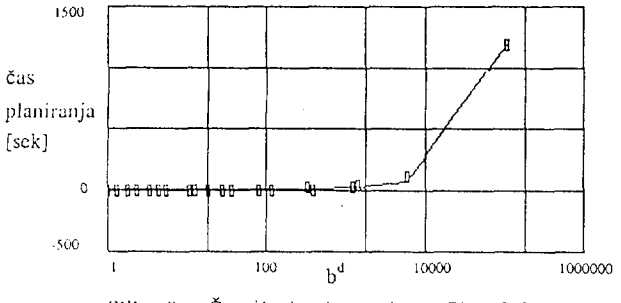
4 REZULTATI

V vzorcu 20-ih problemov planiranja je bilo več različnih problemov, vendar z enakim številom blokov (2, 3, 4 ali 5). Zanje smo izračunali srednjo vrednost dobljenih rezultatov in jih podali v tabelah 5, 6 in 7. Na dno tabel smo uvrstili tudi rezultate problemov planiranja za 10, 20, 30, 50 in 100 blokov. V primeru, da kakšen izmed sistemov ni našel neke rešitve zaradi prevelikih pomnilniških zahtev ali zaradi izjemno dolgega časa planiranja, smo to v tabelah označili s simbolom "-".

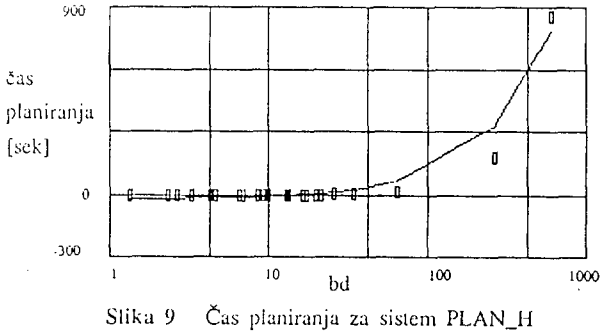
Tabela 5 Primerjava časa planiranja

	sistem			
	PLAN_S d = opt.	PLAN_S d = 2(n-1)	PLAN_H	PLAN_C
n	[min:sec]	[min:sec]	[min:sec]	[min:sec]
1	00:00.00	00:00.00	00:00.00	00:00.00
2	00:00.05	00:00.05	00:00.08	00:00.05
3	00:00.49	00:00.65	00:00.30	00:00.10
4	00:04.94	00:26.92	00:00.77	00:00.16
5	02:57.12	06:04.69	00:01.70	00:00.26
10	-	-	00:14.06	00:00.88
20	-	-	03:02.74	00:04.77
30	-	-	14:18.27	00:13.95
50	-	-	-	01:04.38
100	-	-	-	-

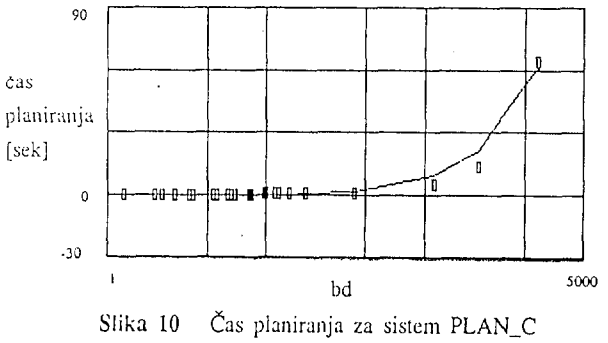
Slike 8, 9 in 10 prikazujejo izmerjene rezultate (pravokotniki) in predpostavljene časovne zahtevnosti (lomljenke), pri čemer je merilo na abscisi logaritemsko. Upoštevali smo vse rezultate eksperimentov in ne več njihovih srednjih vrednosti. Pri sistemu PLAN_S predstavlja diagram odvisnost časa planiranja od vrednosti b^d (slika 8). Odvisnosti časa planiranja od vrednosti bd za sistema PLAN_H in PLAN_C smo prikazali na slikah 9 in 10.



Slika 8 Čas planiranja za sistem PLAN_S



Slika 9 Čas planiranja za sistem PLAN_H



Slika 10 Čas planiranja za sistem PLAN_C

Koliko stanj so obravnavali (razvili ali obiskali) med planiranjem posamezni sistemi, prikazuje tabela 6.

Tabela 6 Primerjava števila obravnavanih stanj

	system			
	PLAN_S d = opt.	PLAN_S d = 2(n-1)	PLAN_H	PLAN_C
n	št. stanj	št. stanj	št. stanj	št. stanj
1	1	1	1	1
2	3.0	3.0	2.5	2.5
3	15.8	20.5	3.5	3.5
4	99.0	524.5	4.8	4.7
5	2181.3	4520.3	7.1	6.1
10	-	-	16	10
20	-	-	53	19
30	-	-	112	28
50	-	-	-	46
100	-	-	-	-

Pri sistemih za reševanje problemov planiranja je ena izmed važnih kvalitet, da sistem najde najkrajši plan. Dolžine najdenih planov za vse sistem smo podali v tabeli 7.

Tabela 7 Primerjava dolžine najdenega plana

	sistem			
	PLAN_S d = opt.	PLAN_S d = 2(n-1)	PLAN_H	PLAN_C
n	dol. plana	dol. plana	dol. plana	dol. plana
1	0	0	0	0
2	1.5	1.5	1.5	1.5
3	2.5	3.0	2.5	2.5
4	3.5	5.3	3.5	3.7
5	4.5	8.0	4.5	5.1
10	-	-	7	9
20	-	-	14	18
30	-	-	21	27
50	-	-	-	45
100	-	-	-	-

5 DISKUSIJA

V tem članku je podana primerjava delovanje treh različnih sistemov (strategij) za planiranje. Analiza učinkovitosti je opravljena na testnih primerih iz blokovnega sveta.

Rezultati eksperimentov se skladajo s teoretičnimi izhodišči, poleg tega pa nudijo tudi povsem praktične napotke, ki so uporabni pri snovanju sistemov za planiranje v prologu.

- Časovna zahtevnost planiranja z iskanjem v globino v sistemu PLAN_S se zelo dobro ujema (korelacijski faktor r = 0.999) s teoretično napovedanim redom $O(b^d)$ (slika 8).

- V nekaterih primerih je problem planiranja s sistemom PLAN_S lažje in hitreje rešljiv z večjo omejitvijo globine kot pa z manjšo. Ta anomalija se pojavi tedaj, ko že od vsega začetka razvijamo plan po pravi poti in ko do vračanja sploh ne pride.

- Učinkovitost sistema PLAN_S lahko povečamo z vgraditvijo dvosmernega iskanja, kot je to realizirano pri sistemih PLAN_H in PLAN_C.

- Rezultati eksperimentov s sistemoma PLAN_H in PLAN_C nakazujejo (r = 0.972 za PLAN_H in r = 0.973 za PLAN_C), da je časovna zahtevnost reda O(bd) (sliki 9 in 10), s čimer nam je omogočeno reševanje večjih problemov, kot s sistemom PLAN_S (tabela 5).

- Pri hevrističnem planiranju igra poleg sprejemljivosti [PEAR84] hevristične ocenitvene funkcije pomembno vlogo njena učinkovita izračunljivost. Za reševanje povprečnega problema planiranja s petimi bloki (tabeli 5 in 6) porabi sistem PLAN_H za obdelavo enega stanja (!) skoraj šestkrat več časa kot sistem PLAN_C in trikrat več časa kot sistem PLAN_S.

- Hevristično planiranje s sprejemljivo hevristično ocenitveno funkcijo in v povezavi z omejitvijo **legal** v sistemu PLAN_H nam da optimalen (najkrajši) plan.

- Obstaja tesna odvisnost časa planiranja od števila obravnavanih stanj. Statistična analiza to nakazuje s korelacijskimi faktorji r > 0.9 za vse tri sisteme.

Na področju planiranja v umetni inteligenci že obstajajo boljše strategije od teh, ki so opisane v tem članku. Omenimo samo nekatere izmed najperspektivnejših:

- razgradnja problemov v **podcilje** ("subgoals"): Rešitev celotnega problema dobimo s kompozicijo delnih rešitev.

- **makro operatorji** ('macro operators'): V problemih skušamo odkriti karakteristične vzorce - strukture, pri katerih smemo uporabiti določene makro operatorje.

- **hierarhično planiranje** ("hierarchical planning"): Probleme rešujemo na večih nivojih s pomočjo abstrakcije.

Za nadaljevanje razvoja sistemov planiranja neodvisnega od domene menimo, da bo potrebno uporabiti kombinacijo večih strategij. Smatramo, da bi bila ena izmed perspektivnih možnosti hierarhični pristop k reševanju problemov, kjer bi na vsakem nivoju lahko uporabljali vnaprej definirane enostavne ali makro operatorje, sama izbira operatorjev pa bi lahko potekala z uporabo učinkovitih hevrističnih ocenitvenih funkcij. Sistemi bi morali biti zgrajeni tako, da bi 'razumeli' primerno formalizirano znanje iz različnih področij, hkrati pa bi morali znati to znanje tudi učinkovito uporabljati.

LITERATURA

[AIGN79] Aigner, M., *Combinatorial Theory*. New York: Springer-Verlag, 1979.

[ALLE90] Allen, J., Hendler, J., and Tate, A. (eds.), *Readings in Planning*. San Mateo, California: Morgan Kaufman, 1990.

[BARR83] Barr, A. and Feigenbaum, E. A., *The Handbook of Artificial Intelligence*, Vol. III. Los Altos, California: William Kaufmann, 1983.

[BRAT86] Bratko, I., *Prolog Programming for Artificial Intelligence*. Wokingham, England: Addison-Wesley, 1986.

[CHAP87] Chapman, D., "Planning for Conjunctive Goals", *Artificial Intelligence*, 32(3): 333- 337, 1987.

[GENE87] Genesereth, M. R. and Nilsson, N. J., *Logical Foundations of Artificial Intelligence*. Los Altos, California: Morgan Kaufman, 1987.

[GEOR90] Georgeff, M. P., "Planning", in Allen, J., Hendler, J., and Tate, A. (eds.), *Readings in Planning*. San Mateo, California: Morgan Kaufman, 1990, pp. 5-25.

[KORF87] Korf, R. E., "Planning as Search: A Quantitative Approach", *Artificial Intelligence*, 33(1): 65-88, 1987.

[NILS90] Nilsson, N. J., "Foreword", in Allen, J., Hendler, J., and Tate, A. (eds.), *Readings in Planning*. San Mateo, California: Morgan Kaufman, 1990, pp. xi-xii.

[PEAR84] Pearl, J., *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Reading, Massachusetts: Addison-Wesley, 1984.

[RAND83] *The Random House Dictionary of the English language*. New York: Random House, 1983.

[SACE74] Sacerdoti, D. E., "Planning in a Hierarchy of Abstraction Spaces", *Artificial Intelligence*, 5(2): 115-135, 1974.

[STEF81a] Stefik, M., "Planning with Constraints (MOLGEN: Part 1)", *Artificial Intelligence*, 16(2): 111-140, 1981.

[STEF81b] Stefik, M., "Planning and Meta-Planning (MOLGEN: Part 2)", *Artificial Intelligence*, 16(2): 141-170, 1981.

[STER86] Sterling, L. and Shapiro, E., *The Art of Prolog: Advanced Programming Techniques*. Cambridge, Massachusetts: MIT Press, 1986.

[TATE90] Tate, A., "A Review of AI Planning techniques", in Allen, J., Hendler, J., and Tate, A. (eds.), *Readings in Planning*. San Mateo, California: Morgan Kaufman, 1990, pp. 26-49.

[VERB87] Verbinc, F., *Slovar tujk*. Ljubljana: Cankarjeva založba, 1987.

[WILK88] Wilkinson, D. E., *Practical Planning: Extending the Classical AI Planning Paradigm*. San Mateo, California: Morgan Kaufman, 1988.

[WINS84] Winston, P. H., *Artificial Intelligence*. Reading, Massachusetts: Addison-Wesley, 1984.

DINAMIČNO DODELJEVANJE PROCESOV NA POLJU TRANSPUTERJEV

INFORMATICA 4/91

Keywords: parallel processor system, token data flow, transputer, topology, simulation, dynamic scheduling

Peter Zaveršek, VELCOM, Velenje
Peter Kolbezen, Institut Jožef Stefan, Ljubljana

V članku je obravnavana problematika dodeljevanja procesov na večprocesorskem vezju, ki ga sestavljajo večkratno povezane zanke. Lokacija procesiranja posameznega procesa ni vnaprej določena. Obravnavani so trije načini dodeljevanja procesov: pri enonivojskem načinu prenosa podatkov po eni zanki ter pri enonivojskem in dvonivojskem načinu prenosa podatkov po dveh zankah hkrati. Za vse tri primere dinamičnega dodeljevanja je izdelan simulator dodeljevanja procesov. S pomočjo simulatorja sta analizirani izkoriščenost polja in hitrost izvajanja algoritma v odvisnosti od parametrov, kot so velikost in konfiguracija procesorskega polja, topologija programskega grafa in časova karakteristika procesov. Na osnovi opravljene analize je predlagan najboljši način dodeljevanja procesov in za različne topologije programskega grafa takšen izbor parametrov, ki vodi k čim večji učinkovitosti večprocesorskega sistema.

PROCESS ALLOCATION IN THE MULTITRANSPUTER NETWORK. The paper deals with a problem of process allocation in a multitransputer network which consists of multiple-connected unidirectional rings. Executing location of every certain process is not explicitly stated, i.e. process allocation is dynamic. Three different possible algorithms for process allocation are presented. The algorithms are as follows: a) one-level communication and sending data in one direction, and b) one-level communication and sending data in two directions simultaneously, c)two-level communication and sending data in two directions simultaneously. Efficiency of different network configurations, program graph topologies and ratio of process execution time to process transfer time were verified by a simulation. Finally the results of simulation are presented and the most efficient process allocation algorithm of the three proposed is selected.

1. UVOD

Zahtevam po vedno večji moči procesiranja je mogoče zadostiti le s paralelnim računanjem na dovolj veliki množici procesorjev. Takšno množico, ki ima paralelno strukturo, lahko predstavlja polje procesorjev. Posameznih procesorjev polja ni mogoče izkoristiti tako dobro, kot pri enem samem procesorju. Procesorji, ki delujejo paralelno, praviloma niso zasedeni ves čas izvajanja algoritma. Izkoriščenost (zasedenost) po procesorjih je neenakomerno porazdeljena. Idealnim razmeram se je mogoče le bolj ali manj približati. Pri tem igra pomembno vlogo več dejavnikov. Pomembnejši med njimi so:

Paralelizacija algoritma. Postopek naj bi odkril ves inherentni paralelizem v danem algoritmu, ki se bo izvajal na paralelnem sistemu. Z ustrezno analizo algoritma ugotovimo, kateri procesi se lahko izvajajo paralelno in kateri sekvenčno, čas izvajanja posameznih procesov in število podatkov, ki so za izvajanje potrebni.

Granulacija algoritma. Algoritem, ki določa neko opravilo in ga izvajamo na našem sistemu, razstavimo na več podopravil. Na koliko podopravil razbijemo opravilo glede na velikost celotnega opravila, kar določa razdrobljenost (granulacijo) algoritma, je odvisno od števila in od zmogljivosti razpoložljivih procesorjev. Podopravilo (subtask ali grain) se avtonomno izvaja na enem od procesorjev polja. Granulacija vpliva na razmerje med časom, ki je potreben za prenos podatkov med dvema procesorjema in časom izvajanja procesa. To razmerje imenujemo količnik prenosnega časa. Izbira najustreznejše granulacije pri danem večprocesorskem sistemu je odvisna od topologije in velikosti procesorskega sistema in ima velik vpliv na učinkovitost izvajanja algoritma. Zato moramo pri pripravi algoritma za izvajanje na paralelnem sistemu granulaciji posvetiti še posebno pozornost.

Dodeljevanje procesov. Procese moramo razporediti po polju procesorjev tako, da je polje čimbolj izkoriščeno. Pri tem je pomembno, da je število komunikacij med procesi čimmanjše in čas dodeljevanja čimkrajši.

Porazdelitev bremena. Procesorji v polju morajo biti enakomerno obremenjeni. Neenakomerna obremenjenost zmanjša učinkovitost polja in poveča ceno, ki je posledica povečanega skupnega časa procesiranja.

V delu je obravnavana problematika, ki je prisotna predvsem v zadnjih dveh dejavnikih, in se opira na simulacijo predlaganega večprocesorskega sistema. Sistem je sestavljen iz polja transputerjev in je podrobneje opisan v delu /6/. Zanj so značilni:

Krožna konfiguracija: Konfiguracijo sistema sestavlja večje število med seboj prepletenih prstanov (linkov), ki povezujejo vse transputerje polja. Oblika poti ni pomembna. Pot mora biti le krožno zaključena neglede na to, kje v sistemu ima izhodiščno točko. Primer takšnega sistema je hiperkocka.

Hierarhično dodeljevanje procesorjev: Polje procesorjev izvaja podopravila. V podopravilu se operacije (imenovane procesi) izvajajo, kolikor je mogoče konkurenčno. Procesi so v značilnem odnosu, toda neodvisni med seboj (tako kot DO zanke v fortranu). Podopravila je mogoče predstaviti v obliki hierarhičnih acikličnih grafov pretoka podatkov (HADFG). Vozlišče takšnega podgrafa je proces, ki se izvaja na enem od transputerjev polja. Znotraj vsakega procesa so možne zanke. Kontrolo nad podopravili, ki so določeni z eno ali več HADFG, ima poseben procesor na vhodu v procesorsko polje.

Dinamično dodeljevanje procesorjev: Razvrščanje procesov je avtomatizirano in dinamično. Dosledno sledi pravilu, ki ga določa položaj procesa ali veje v strukturi HADFG.

Komunikacija s pomočjo žetonov: Komunikacijski mehanizmi so osnovani na uporabi žetonov, ki krožno potujejo znotraj posameznih prstanov. Komunikacije so enosmerne. Fizični prstan sestavljata dva komunikacijska prstana. Eden od njiju je uporabljen za prenos krmilnih žetonov, drugi pa za prenos podatkov, programskih blokov in rezultatov.

2. DODELJEVANJE PROCESOV

Poznanih je več načinov dodeljevanja procesov. Bistveno se med seboj razlikujejo predvsem načini statičnega in dinamičnega razporejanja procesov.

Pri statičnem razporejanju preslikamo posamezne procese na določene procesorje že pred začetkom izvajanja. Preslikava je osnovana na analizi sočasnosti in čimboljši izkoriščenosti procesorjev. Praviloma je pri statičnem razporejanju mogoče doseči večjo izkoriščenost sistema.

Pri dinamičnem razporejanju se preslikajo posamezni procesi na procesorje med samim izvajanjem algoritma. Izkoriščenost procesorjev, ki mora biti v vsakem trenutku čimboljša, se dinamično spreminja in je odvisna od vsakokratnega trenutnega stanja sistema.

Uporaba dinamičnega načina razporejanja je posebej primerna v naslednjih primerih:

- čas izvajanja procesov ni vnaprej določen,
- v fazi analize algoritma ne poznamo konfiguracije sistema, na katerem se bo algoritem izvajal,
- želimo trdoživ (fault tolerant) sistem.

2.1. DINAMIČNO RAZPOREJANJE PROCESOV

Sistemi z dinamičnim dodeljevanjem procesov ugotavljajo trenutno stanje sistema na več načinov /7/:

- **Bolj obremenjen procesor išče manj obremenjenega.** Ta način je uspešnejši za malo in srednje obremenjene procesorje.
- **Manj obremenjen procesor išče bolj obremenjenega.** Rešitev je bolj primerna za zelo obremenjene procesorje.

Naš sistem uporablja oba zgoraj omenjena načina. Obremenjeni procesor išče nezaseden procesor, da bi mu predal enega izmed čakajočih procesov. Ko ga najde, mu preko sporočila preda podatke o procesu, ki je prvi na vrsti na čakajoči lestvici procesov. Postopek se ponavlja, dokler procesor ne ugotovi, da ni več nezasedenega procesorja. Takrat se postavi v stanje čakanja. Procesor, ki zaključi svoj proces in se tako sprosti, pošje procesorjem v zanki sporočilo, da ni več zaseden. Tedaj mu pošlje procesor, ki ima še čakajoče procese, zahteve oz. podatke o naslednjem procesu, ki ga bo izvajal in cikel se tako ponovi /6/.

2.2. TEHNIKA ZASEGANJA VIROV IN ZASEDENOST SISTEMA

Poznanih je več tehnik zaseganja sistemskih virov. Zlasti sistemi hierarhične zgradbe imajo lahko poseben nadzorni procesor, ki hrani informacijo o stanju vseh ostalih (delovnih) procesorjev. Nadzor nad zasedenostjo procesorjev je osredotočen na enem samem mestu. Slabost takega načina so dodatne komunikacije, ki so potrebne zaradi nenehnega testiranja stanj sistema. Ker so podatki o sistemu shranjeni v skupni tabeli, do katere imajo dostop vsi procesorji, obstaja možnost sočasnega dostopa in

lahko pride do konfliktne situacije. V primeru, da ima vsak procesor svojo kopijo tabele, ki vsebuje stanja vseh procesorjev sistema, je nadzor nad zasedenostjo procesorjev enako porazdeljen na vse procesorje. Pri tem je potrebno vse kopije sproti osveževati. Možnost konfliktnih situacij je v tem primeru lahko ob primerni organiziranosti sistema manjša, z večanjem števila procesorjev v sistemu pa se hitro veča tudi število komunikacij.

V obravnavanem sistemu je izdelana posebna tehnika zaseganja sistemskih virov, pri kateri je nadzor nad zasedenostjo procesorjev porazdeljen le nekaterim procesorjem, t.im. korenskim procesorjem v sistemu. Ti imajo evidenco o zasedenosti tistih procesorjev, ki so korenskemu procesorju dosegljivi.

2.3. PRENAŠANJE SPOROČIL

Način prenašanja sporočil ima pomembno vlogo v večprocesorskem sistemu. Sporočilo je sestavljeno iz niza elementarnih podatkovnih paketov FCD (imenovanih "flits" - Flow Control Digits). FCD je najmanjša enota sporočila, ki jo lahko kanal ali vrsta sprejme ali zavrne. Večja hitrost prenosa preko večih posrednikov (procesorjev v sistemu) je dosežena tako, da vsak posrednik prične predajati sporočilo naslednjemu procesorju, še predno ga sprejme v celoti. Predaja prvega FCD se začne takoj, ko procesor preveri, da sporočilo ni namenjeno njemu. Posamezne FCD sporočila pa lahko začasno tudi zadrži, če je pot naprej zasedena. Takšno prenašanje sporočila, ki leze po sistemu kot črv, je v tuji literaturi poznano pod imenom "cut-through" /8/.

Pri prenosu sporočila ne sme nastopiti smrtni objem (deadlock) ali živi objem (livelock). Smrtnemu in živemu objemu se v našem transputerskem sistemu izognemo že na dokaj enostaven način. Smrtni objem preprečimo z dovolj velikimi vmesniki za prenos sporočil pri prenosnih procesorjih /2,8/, živi objem pa z enosmernimi zankami.

3. MEDTRANSPUTERSKE KOMUNIKACIJE

Transputer uvrščamo v družino RISC procesorjev /5/. Na tržišču je že več transputerjev, ki se med seboj razlikujejo predvsem po svoji zmogljivosti (hitrosti), protokolu komuniciranja, podpori za aritmetične operacije in pd., lahko pa tudi po namembnosti, ki jo določi že proizvajalec. Za nas je pomembna značilnost, da transputer že na strojnem nivoju podpira paralelno izvajanje procesov (v določeni meri kvaziparalelno). Komunikaciji so namenjene štiri vgrajene dvosmerne zanke (linki). Te zanke omogočajo, da lako transputerje medsebojno povezujemo brez dodatnih vmesnikov. Komunikacija preko zanke lahko v obeh smereh poteka istočasno, avtomatično in skoraj brez obremenjevanja transputerja. Uporabniku je zunanja komunikacija povsem identična komunikaciji med notranjimi procesi.

Prenos podatkov med transputerji je paketni in poteka preko komunikacijske zanke s hitrostjo, ki je po tovarniških podatkih 5, 10 ali 20 Mb/s. Tej vrednosti se lahko približamo, če imamo prenos v eni zanki in le v eni smeri, in če transputer ne opravlja nobenih drugih opravil. Realno dosežena hitrost prenosa je v veliki meri odvisna tudi od vrste uporabljenega transputerja.

Prenosni paket sestavlja 11 bitov:

1 bit	startni bit
1 bit	zastavica "podatki"
8 bitov	podatek
1 bit	stop bit

Transputer, ki sporočilo sprejema, mora potrditi vsako besedo sporočila posebej. Potrditveno sporočilo je dolgo 2 bita:

1 bit	startni bit
1 bit	stop bit

Transputer T414 /4/ prenaša sporočilo tako, da najprej sprejme celo besedo sporočila (11 bitov) in nato odda potrdilo o sprejemu (2 bita). Teoretična hitrost prenosa je:

$$CT_{\text{T414}}^{\text{uni}} = \frac{20Mbs^{-1}}{13ciklov} = 1{,}54Mbs^{-1}$$

Transputer T800 ima izboljšan protokol prenosa. Ne čaka na sprejem cele besede sporočila, temveč potrdi sprejem sporočila že pred zaključkom prenosa. Oddana beseda in potrditev sprejema se med seboj prekrivata. Zato je za prenos sporočila potrebno manjše število ciklov. Dolžina sporočila je le 11 bitov:

$$CT_{\text{T800}}^{\text{uni}} = \frac{20Mbs^{-1}}{11ciklov} = 1{,}82Mbs^{-1}$$

Dvosmerni prenos preko ene zanke zahteva dodatne potrditvene bite za nasprotno smer, kar pomeni za T800 13 bitov in prenos s hitrostjo $CT = 1{,}54MBs^{-1}$. To je teoretična hitrost prenosa za vsako zanko (link) v vsaki smeri. Teoretična skupna možna hitrost

prenosa preko štirih dvosmernih zank je $12MBs^{-1}$, v eni smeri pa je potem okoli $1,5MBs^{-1}$.

Realna slika je seveda drugačna. Čeprav delujejo komunikacijske zanke ločeno in so ločene tudi od ostalih procesov, ki se izvajajo na transputerju, prihaja do medsebojnih vplivov. Realne vrednosti so zato dosti manjše od teoretičnih.

Prenosi preko zank (sprejem, oddaja, vmesno shranjevanje sporočil) zahtevajo dostop do transputerjevega pomnilnika. Prenosi do pomnilnika potekajo v obliki DMA prenosov in so načelno med seboj neodvisni, vendar zasedejo določen del pasovne širine transputerjevega podatkovnega in naslovnega vodila. Pasovna širina podatkovnega vodila znaša za notranji pomnilnik $80MBs^{-1}$, za zunanji pomnilnik pa $26.6MBs^{-1}$. Primerjava s teoretično najvišjo možno hitrostjo prenosa nam pokaže, da to pomeni 15% pasovne širine vodila notranjega pomnilnika oz. skoraj 50% pasovne širine vodila zunanjega pomnilnika. Posledice so dvojne. Konfliktne situacije po eni strani zmanjšujejo hitrost prenosov po zankah, po drugi strani pa ovirajo izvajanje samega procesa na transputerju, ki prav tako potrebuje dostop do pomnilnika. Realno so prenosi po zankah redkejši in je zato pogostost konfliktnih situacij manjša.

Procesi, ki izvajajo servisne rutine za prenose, in preklopi med delovnimi procesi (konteksti) nimajo zanemarljivega vpliva na učinkovitost procesiranja. Servisnih procesov je lahko več hkrati in običajno tečejo v prioritetnem načinu (PRI PAR, PRI ALT). Pogostost takšnih procesov lahko občutno podaljša čas izvajanja delovnega procesa in s tem delovno zmogljivost procesorja.

Servisne rutine za prenos podatkov morajo biti čimbolj učinkovite. Temu pripomore predvsem hitrost izvajanja teh rutin. Tudi promet podatkov naj bo redkejši, da je obremenjenost notranjih vodil manjša.

Analiza prenosa podatkov med transputerji in raziskava vpliva prenosa podatkov na učinkovitost sistema, sta opisana v delu /1/. Raziskava je bila omejena na transputer T414 pri taktu 15MHz in pri različnih konfiguracijah transputerskih povezav (prstan, drevesna struktura, hiperkocka). Za vsako konfiguracijo je bila napisana posebna servisna komunikacijska rutina. Testiranje je bilo opravljeno pri različnih dolžinah sporočil v razponu od enega do 1024 bytov.

Rezultati kažejo realen vpliv naštetih faktorjev na zmanjšanje delovne moči transputerja. Enosmerni prenos preko ene same zanke zmanjša moč procesiranja za 5 do 10%. Zanimivo je, da se minimalna moč ne pojavlja na gornji ali spodnji meji dolžine sporočila, temveč pri 64 bytih. Možna razlaga tega pojava je, da prihaja do časovno izredno neugodnega preklapljanja kontekstov.

Istočasni prenos sporočil po različnih zankah in tudi takih, ki dopuščajo dvosmerne prenose, dajejo približno enake rezultate. To velja le za prenose daljših podatkovnih blokov (okoli 1kB). Zmanjšanje delovne moči pri takšnih prenosih je okoli 12%. Krajša sporočila zahtevajo pogostejše preklope na servisni proces in delovna moč transputerja občutno pade - tudi za 85% in več. Zato kratka sporočila niso zaželena.

4. SIMULACIJA PROCESIRANJA

Predpostavljmo, da so podalgoritmi, ki se izvajajo na našem transputerskem polju večprocesorskega sistema, predstavljivi v obliki hierarhičnih acikličnih grafov pretoka podatkov (HADFG). To so drevesni grafi, katerih veje so razvrščene v več nivojev. Vsaka veja se končuje z novim razvejiščem ali s procesom. Nivoji z množicami paralelnih procesov in nivoji z množicami zaporednih procesov se med seboj izmenično prepletajo. Iz dela /6/ je razvidno, da taka predstavitev omogoča avtomatizirano dinamično porazdeljevanje procesov algoritma na polje procesorjev.

Dodeljevanje procesov smo opazovali na simulatorju, ki smo ga posebej zgradili v ta namen z nalogo, da čimbolj verno simulira izvajanje algoritma na našem procesorskem polju. Simulator upošteva zmanjšano delovno moč procesorja, ki je posledica prenosov. Predpostavljamo, da prenos v eni sami smeri zmanjša delovno moč transputerja na 90%, prenos v dveh smereh pa na 85%. Ta ocena je slabša od podane v literaturi /1/.

Simulacija dodeljevanja in izvajanja procesov naj bi dala odgovore na nekatera odprta vprašanja, ki so bila zastavljena že v delu /6/. Predvsem sta nas zanimala:

- najustreznejša granulacija algoritma (količnik prenosnega časa), in
- najustreznejša topologija (prstan, kvadrat, pravokotnik).

4.1. KONFLIKTNE SITUACIJE

V realnem večprocesorskem sistemu nastopajo konfliktne situacije, ki so pogojene z zgodovino izvajanja opravila. Simulator, ki bi simuliral delovanje transputerskega polja do vseh potankosti, bi imel veliko časovno in programsko kompleksnost. Ker smo želeli kompleksnost simulacije zmanjšati, smo jo poenostavili s hevrističnim razreševanjem konfliktov.

Če se pri simulaciji pojavi konfliktna situacija, izberemo enega izmed udeležencev in mu damo prednost pred ostalimi. Posledica tako poenostavljenega razreševanja konfliktov je, da nam lahko dajejo ponavljajoče se simulacije istega HADFG na istem sistemu različne rezultate. Zato smo predpostavili, da predstavlja pravilen rezultat povprečje rezultatov večjega števila simulacij.

4.2. NAČINI RAZPOREJANJA PROCESOV

Uporabili smo več različnih načinov razporejanja procesov. Vsi trije načini so osnovani na enakem osnovnem principu, ki je podrobneje opisan v delu /6/, in se opira na hierarhično drevesno strukturo grafa.

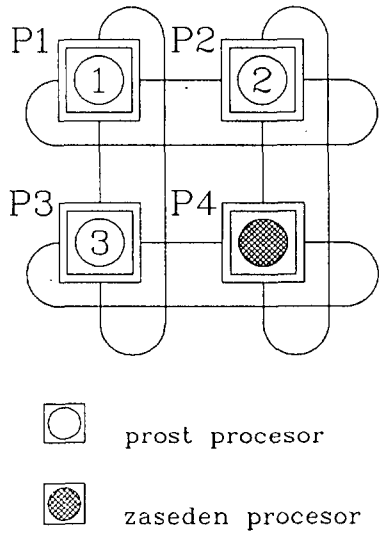
Razporejanje procesov se lahko izvaja v največ dveh nivojih. Procesorji v obeh zankah, ki pripadata začetnemu korenskemu procesorju tvorijo t.i. prvi procesorski nivo. Če je iskanje prostih procesorjev na tem nivoju neuspešno, nadaljujemo iskanje prostih procesorjev na drugem nivoju, na katerem pregledamo še preostali del ravninskega polja. Pregledovanje poteka tako, da pošlje korenski procesor procesorjem prvega nivoja v eni izmed dveh možnih zank (t.im. horizontalni ali vertikalni) zahtevo, da v zankah, ki so pravokotne na prvi nivo najdejo prost procesor. Oglejmo si posamezne načine razporejanja procesov:

a) Enonivojski način razporejanja s prenosom podatkov po eni zanki. Ta način je najpreprostejši. Osnovno idejo razporejanja najlažje predočimo s pomočjo paralelnega razvejišča grafa. Ko se spustimo po grafu do paralelnega razvejišča, zasedemo največ toliko prostih procesorjev, kolikor je vej v razvejišču. Dostopni procesorji se nahajajo neposredno v eni izmed dveh zank prvega procesorskega nivoja. Če pri zaseganju procesorjev zmanjka prostih procesorjev v prvi zanki, jih zasežemo še v drugi. Pri tem se lahko primeri, da je število procesov večje kot število dostopnih procesorjev. V takšnem primeru postavimo čakajoče procese v čakalne vrste. Ko se kateri izmed dostopnih procesorjev sprosti, mu dodelimo naslednji čakajoči proces.

Opisana metoda ima vsaj dve slabosti:

- Dodeljevanje procesov poteka v vsakem trenutku le v eni zanki (horizontalni ali vertikalni), čeprav lahko transputer hkrati predaja sporočila v več smereh.
- Ko pridemo v novo razvejišče, v fazi "priprave" takoj zasežemo vse razpoložljive procesorje. Zaseženi procesorji nadalje čakajo na sprejem podatkov o procesu, ki ga morajo izvajati. V stanju čakanja so blokirani in neuporabni, kar zmanjšuje izkoriščenost procesorskega polja.

b) Enonivojski način razporejanja s prenosom podatkov po dveh zankah hkrati. Posebnost tega načina je, da pri razporejanju ne rezerviramo vnaprej vseh dostopnih procesorjev. Izberemo le en prost procesor v vodoravni in le en prost procesor v navpični zanki. Prenos procesov lahko poteka hkrati v obeh smereh. Takoj, ko se eno od sporočil prenese, poiščemo naslednji prost procesor v isti smeri, tj. v smeri že prenešenega sporočila. Tako se lahko tudi faza "priprave" v eni smeri in faza prenosa sporočila v drugi smeri izvajata hkrati. To se dogaja v primerih, ko sta dolžini sporočil različni. V primerjavi z zgornjim načinom razporejanja so lastnosti tega načina naslednje:



Slika 1. Neizkoriščenost polja pri enonivojskem dodeljevanju

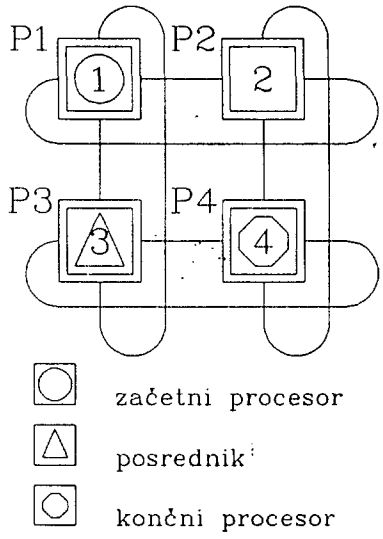
- Sistem je hitrejši zaradi istočasnega prenašanja sporočil v obeh smereh.
- Blokiranost procesorjev v polju je manjša.
- Delovna moč korenskega procesorja je zaradi povečanega obsega strežbe manjša.

Skupna slabost obeh gornjih načinov razporejanja je, da lahko paralelne procese dodeljujeta le v vodoravni in navpični zanki glede na korenski procesor, tj. le na prvem procesorskem nivoju. Večjega števila procesov, kot je prostih procesorjev v teh dveh zankah, korenski procesor ne more oddati. Zato obstaja možnost, da ostanejo nekateri procesorji polja neizkoriščeni.

Slika 1 kaže neučinkovitost enonivojskega razporejanja na izkoriščenost sistema. Na polju štirih transputerjev se lahko izvajajo štirje procesi hkrati, v našem primeru pa le trije. Četrti proces čaka na

procesor toliko časa, dokler se ne sprosti eden od treh že zasedenih procesorjev, medtem ko četrti procesor ostane ves čas neizkoriščen.

c) Dvonivojski način razporejanja pri prenosu podatkov po dveh zankah hkrati. Ta način omogoča, da so lahko hkrati izkoriščeni vsi procesorji polja. Za razliko od gornjih dveh metod iščemo tu nezasedene procesoje po celotnem procesorskem polju in ne le



Slika 2. Izkoriščenost polja pri dvonivojskem dodeljevanju

po zankah, ki potekata skozi začetni korenski procesor. Na prvem nivoju iskanja prostih procesorjev je način delovanja enak, kot pri prvih dveh načinih. Če je iskanje na tem nivoju neuspešno, ga nadaljujemo na drugem nivoju in tako pregledamo celotno ravninsko polje procesorjev. Iskanje prostih procesorjev na drugem nivoju poteka takole:

Začetni (korenski) procesor najpreje ugotovi zanko v katero ne oddaja nobenega procesa. Procesorji v izbrani zanki so procesorji prvega nivoja. Tem procesorjem pošlje žeton z zahtevo, da mu najdejo prost procesor na drugem nivoju. Procesorji, ki so v zanki, iščejo prost procesor drug za drugim. Procesor, ki je našel prost procesor, postane posrednik pri prenosu podatkov in sporočil.

Slika 2 kaže pri uporabi tretjega načina razdeljevanja procesov na danem primeru popolno izkoriščenost procesorskega polja. Ta način se je izkazal kot najboljši.

4.3. SIMULACIJA IZVAJANJA HADFG

Najpreje smo generirali množico G_n 20 naključnih grafov HADFG. Vseh 20 grafov smo obravnavali kot sortirane in nesortirane grafe. Vsak graf iz množice G_n smo tudi sortirali in tako dobili množico sortiranih grafov G_s. Skupaj smo simulirali izvajanje 2 x 20 grafov iz mnžice G, $\left(G = G_n \cup G_s\right)$. Pomen sortiranja grafov in definicijo sortiranega grafa najdemo v delu /6/.

Izvajanje vsakega grafa smo simulirali na različnih konfiguracijah procesorskega polja s številom procesorjev od 1 do 16. Izbirali smo konfiguracije iz nabora konfiguracij $K = a \times b$, kjer sta $a,b \in \{1,2,3,4\}$.

V prvi fazi raziskave smo opazovali čas izvajanja grafa v odvisnosti od izbranih konfiguracij. Na osnovi dobljenih rezultatov smo se pri nadaljnih raziskavah omejili na tiste konfiguracije polja, pri katerih so se pokazali boljši rezultati.

V drugi fazi raziskave smo opazovali čas izvajanja grafa HADFG v odvisnosti od količnika prenosnega časa. Količnik smo spreminjali po stopnjah 1, 2, 5, 10, 20, 50, 100, 200, 500, 1000. Pri tej raziskavi smo izbrali graf iz tiste podmnožice G^* grafov množice G, za katero velja:
1., da je $G^* \gg \rceil\ G^*$, in
2., da so grafi iz množice G^* v prvi fazi raziskave dali podobne rezultate.

Tretjo fazo raziskave smo posvetili izbiri najprimernejšega načina razporejanja procesov pri **dopustnem količniku prenosnega časa** v odvisnosti od dveh kriterijev:

- časa izvajanja HADFG
- učinkovitosti procesorskega polja

4.3.1. DOPUSTNI KOLIČNIK PRENOSNEGA ČASA

Pri dani konfiguraciji polja je čas izvajanja grafa HADFG poleg algoritmičnih lastnosti grafa HADFG tudi funkcija razmerja T_{izv}/T_p. Normirani čas izvajanja grafa HADFG je razmerje med tem časom pri izbranem razmerju T_{izv}/T_p in med minimalnim časom izvajanja.

Dopustni količnik prenosnega časa je tisto razmerje T_{izv}/T_p, pri katerem je normirani čas izvajanja grafa enak dvakratni vrednosti minimalnega časa izvajanja.

4.3.2. ČAS IZVAJANJA HADFG

Čas izvajanja grafa HADFG smo vzeli kot merilo za primerjavo med različnimi načini razporejanja. Ker je ta čas odvisen od algoritmičnih lastnosti grafa HADFG in od konfiguracije procesorskega polja, smo definirali vsoto $S_k\left(G_j\right)$, ki je vsota časov iz-

vajanj HADFG iz izbrane množice algoritmov pri določeni konfiguraciji procesorskega polja. Velja:

$$S_k\left(G_j\right)=\sum_i T_i$$

kjer je:

k ... element iz množice konfiguracij {2x2, 3x3, 4x4}
G_j ... element iz množice $\{G_s, G_n\}$
T_i ... čas izvajanja i -tega grafa HADFG iz množice G_j. V našem primeru je i element iz množice $\{1,2,...,20\}$.

Primerjava vsot $S_k\left(G_j\right)$ pri različnih načinih razporejanja nam daje oceno uspešnosti posameznih načinov.

4.3.3. UČINKOVITOST PROCESORSKEGA POLJA

Učinkovitost procesorskega polja E je razmerje med ceno izvajanja grafa HADFG na enem procesorju in ceno izvajanja na večprocesorskem sistemu. Idealna vrednost je 1, kar pomeni, da je paralelni sistem maksimalno izkoriščen.

$$E=\frac{T_s}{n\cdot T'}$$

kjer je:

n ... število procesorjev v sistemu
T' ... povprečen čas zasedenosti procesorjev
T_s ... čas sekvenčnega izvajanja grafa HADFG.

Čas sekvenčnega izvajanja grafa HADFG je tisti čas, ki je potreben, da se graf izvede na enem samem procesorju. Torej je:

$$T_s=\sum_i T_{pi},$$

kjer je T_{pi} čas izvajanja i-tega procesa grafa HADFG.

4.3.4. ZASEDENOST PROCESORSKEGA POLJA

Procesor je zaseden v dveh primerih:

- kadar izvaja proces, ki je določen z vozliščem grafa
- kadar pričakuje podatke ali rezultate (je blokiran)

Naj bo T_{izvj} čas izvajanja procesov na i-tem procesorju in T_{blokj} čas blokiranosti i-tega procesorja polja.

Čas zasedenosti procesorskega polja T_{zas} je podan z enačbo

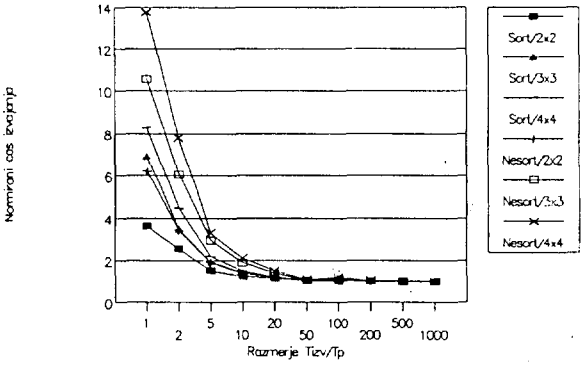
$$T_{zas}=\sum_i T_{izvi}+\sum_i T_{blokj}$$

Zasedenost procesorskega polja Z je razmerje med časom povprečne zasedenosti procesorjev v polju in časom izvajanja grafa HADFG na tem istem polju.

$$Z=\frac{T'_{zas}}{T_{graf}},$$

kjer je:

T_{graf} ... čas izvajanja grafa HADFG na procesorskem polju, in
T'_{zas} ... čas povprečne zasedenosti procesorjev v polju



Slika 3. Normirani čas izvajanja za različne K

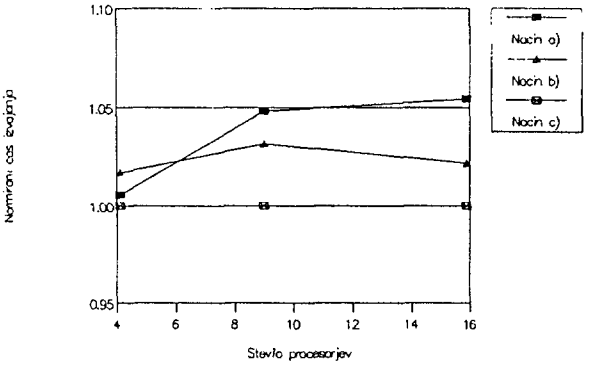
Povprečna zasedenost procesorjev v polju je:

$$T'_{zas}=\frac{T_{zas}}{n},$$

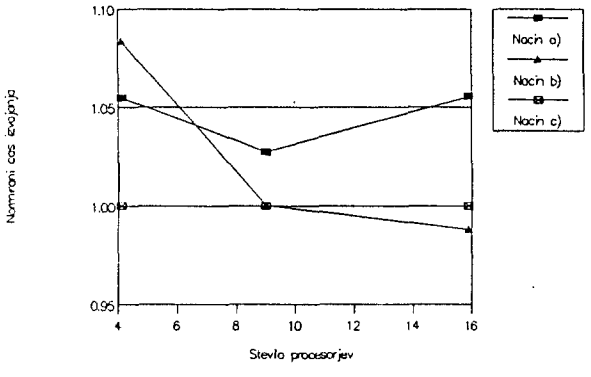
kjer je T_{zas} čas zasedenosti procesorskega polja in n število procesorjev.

5. REZULTATI SIMULACIJE

Simulacija je pokazala, da je najbolj primerna kvadratna oblika procesorskega polja. Takšna oblika omogoča v povprečju krajše razdalje med procesorji in manjšo obremenitev prenosnih poti. Zato smo se pri nadaljnih raziskavah omejili na kvadratna procesorska polja velikosti 2x2, 3x3 in 4x4.

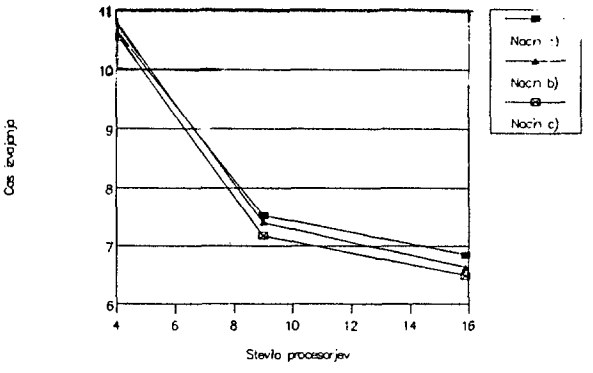


a) sortirani grafi

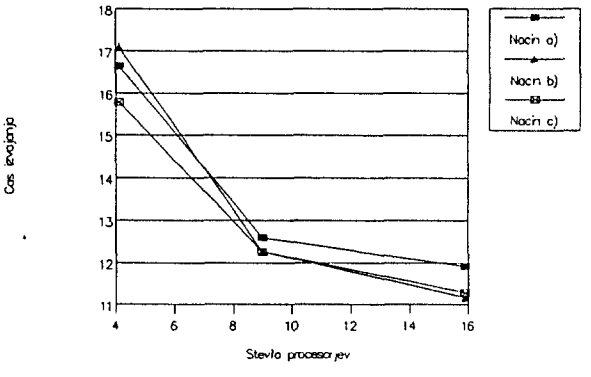


b) nesortirani grafi

Slika 4. Primerjava po normiranem času izvajanja



a) sortirani grafi

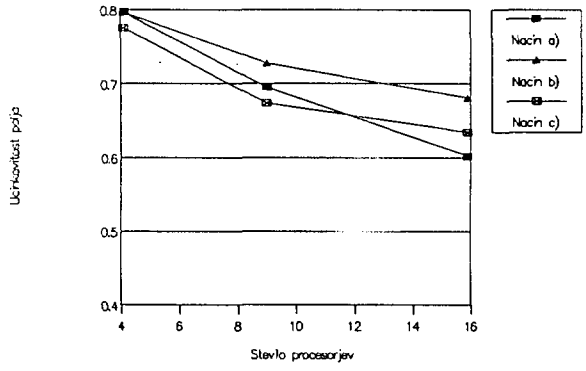


b) nesortirani grafi

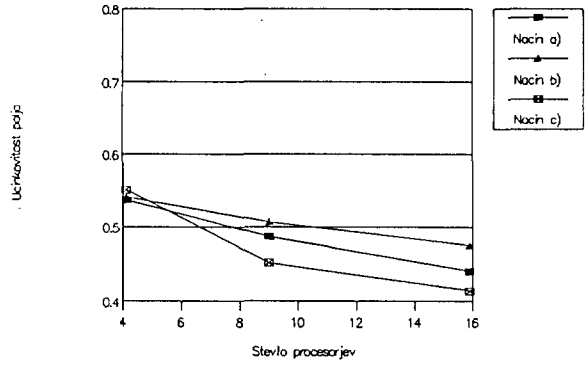
Slika 5. Primerjava po času izvajanja

Iz slike 3 je razvidno, da se dopustni količnik prenosnega časa spreminja z velikostjo procesorskega polja. Manjše procesorsko polje zahteva nižji, večje polje pa višji dopustni količnik časa prenosa. Dopustni količnik za polje 4x4 za sortiran graf je 6 in za nesortiran graf je 12. Pri nadaljnih simulacijah smo se omejili na količnik 10.

Slika 4 prikazuje normirano vsoto, ki je podana z razmerjem med $T_k\left(G_j\right)$ pri izbranem načinu razporejanja in $T_k\left(G_j\right)$ pri dvonivojskem načinu razporejanja, v odvisnosti od konfiguracije procesorskega polja. Dvonivojski način razporejanja daje v pogledu časa izvajanja najboljše rezultate za konfiguraciji 2x2 in 3x3. Majhno število procesorjev na prvem nivoju razporejanja preprečuje pri enonivojskem načinu širjenje procesov po celotnem polju, medtem ko lahko dvonivojski način razporejanja uporabi vse nezasedene procesorje. Zanimivo je, da na polju 2x2 kaže najslabše rezultate enonivojski način razporejanja z dvostranskim prenosom. Predvidevamo, da so takšni rezultati posledica dodatnega zmanjšanja moči procesiranja zaradi istočasnih dvostranskih prenosov. Dvonivojsko razporejanje pri konfiguraciji 4x4 in nesortiranih grafih daje presenetljivo slab rezultat, kar je posledica dvonivojskega razdeljevanja večjih poddreves.



a) sortirani grafi



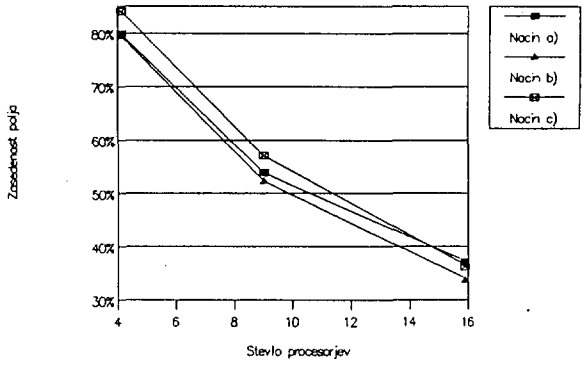
b) nesortirani grafi

Slika 6. Primerjava po učinkovitosti polja

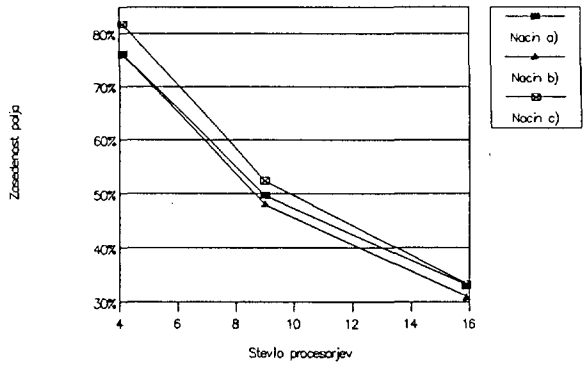
Sortirani grafi imajo paralelne veje razporejene tako, da se veje, ki zahtevajo daljše prenosne čase, izvajajo na bližjih procesorjih, kar občutno zmanjša prenosne čase. Primerjava med vsotami T_k za sortirane in nesortirane grafe na sliki 5 kaže, da se sortirani grafi izvajajo neprimerno hitreje od nesortiranih.

Učinkovitost polja kaže slika 6. Enonivojski način razporejanja s prenosom podatkov po dveh zankah je najbolj učinkovit, dvonivojski način pa najmanj. Izjemoma je pri dvonivojskem načinu razporejanja, nesortiranih grafih, in konfiguraciji 2x2 učinkovitost polja večja.

Zasedenost polja kaže slika 7. Največja zasedenost se pojavlja pri dvonivojskem in najmanjša pri enonivojskem načinu razporejanja s prenosom podatkov po dveh zankah hkrati.



a) sortirani grafi



b) nesortirani grafi

Slika 7. Primerjava po zasedenosti polja

6. ZAKLJUČEK

Članek predstavlja poskus verifikacije hierarhičnega večprocesorskega sistema, ki je opisan v delu /6/. Verifikacija se opira na simulacijo sistema. Izdelana je analiza delovanja transputerskega polja v odvisnosti od načina razporejanja, velikosti polja in količnika prenosnega časa.

Potrjeno je dejstvo, da imajo razdalje med procesorji pomembno vlogo pri prenašanju sporočil. Analiza

sistema je pokazala, da je mogoče doseči s krajšimi zankami in večjim številom možnih komunikacijskih poti boljše rezultate. Zato je kvadratna oblika procesorskega polja praviloma boljša od pravokotne oblike.

Količnik prenosnega časa naj bo za uspešno izvajanje čim večji, medtem ko je velikost dopustnega količnika ocenjena na 10. Ta količnik pogojuje stopnjo granulacije algoritma, ki se izvaja na transputerskem sistemu. Pri količniku, ki je večji od dopustnega, postane učinkovitost sistema bistveno večja.

Obravnavani sistem omogoča več načinov dinamičnega dodeljevanaja procesov. Raziskave učinkovitosti treh predlaganih načinov so dale naslednje rezultate:

- Enonivojski način razporejanja procesov s prenosom podatkov po eni zanki je najmanj učinkovit; je najpočasnejši in slabo izkorišča procesorsko polje.
- Enonivojski način razporejanja procesov s prenosom podatkov po dveh zankah hkrati ima največjo učinkovitost in najbolje izkorišča dano procesorsko polje, vendar ni tudi najhitrejši. Priporočljiv je v primerih, ko izvajamo na enem procesorskem polju več opravil hkrati in v polje vstopamo sočasno skozi več procesorjev. Izkoristek procesorskega časa je tu najboljši.
- Najhitrejše je izvajanje opravil pri dvonivojskem načinu razporejanja procesov s prenosom podatkov po dveh zankah. Cena, ki jo zahteva večja hitrost, se kaže v manjši učinkovitosti procesorskega polja.

7. LITERATURA

/1/ L.C.Waring, A general purpose communications shell for a network of transputers, North-Holland, Microprocessing and Microprogramming Vol.29 (1990), 107-119

/2/ A.Ciampolini, A.Corradi, L.Leonardi, Parallel object system support on transputer-based architectures, North-Holland, Microprocessing and Microprogramming Vol.27 (1989), 339-346

/3/ C.Jesshope, Parallel processing, the transputer and the future, Butterworth & Co. (Publishers) Ltd., Microprocessors and Microsystems, Vol.13 No.1, 1989, 33-37

/4/ INMOS Spectrum, "Product information, The Transputer Family", March 1988.

/5/ C.Jesshope, Transputers and switches as objects in OCCAM, North-Holland, Parallel Computing 8 (1988), 19-30

/6/ P.Kolbezen, P.Zaveršek, Hierarhični večprocesorski sistem, Informatica, A Journal of Computing and Informatics, Vol.15, Nr.1, Feb. 1991, 65-76

/7/ C.Barmon, M.N.Faruqui, G.P.Battacharjee, Dynamic load balancing algorithm in a distributed system, North-Holland, Microprocessing and Microprogramming, Vol.29 (1990), 273-285

/8/ U.De Carlini, U.Villano, The routing problem in transputer-based parallel systems, Butterworth-Heinemann Ltd., Microprocessors and Microsystems, Vol.15 No.1, 1991, 21-33

OCENJEVANO UČENJE V NEVRONSKIH MREŽAH

INFORMATICA 4/91

Andrej Dobnikar, Jelena Ficzko, Mira Trebar
Fakulteta za elektrotehniko in računalništvo

Keywords: reinforcement learning, stochastic reinforcement learning, neural networks

POVZETEK

Ena izmed kategorij učenja v umetnih nevronskih mrežah je ocenjevano (reinforcement, graded) učenje. To vrsto učenja srečamo tudi v bioloških sistemih. V članku je predstavljen stohastični algoritem z ocenjevanim učenjem. Obnašanje algoritma je podano primerjalno za eno samo stohastično enoto, za stohastično enoto v povezavi z back-propagation enoto ter za samo back-propagation enoto, kjer pa je uporabljeno nadzorovano učenje. Čeprav je nadzorovano učenje bistveno hitrejše od ocenjevanega, pa je ta vrsta učenja uporabna tudi v primerih, ko želeni izhodi za vsak vhod niso poznani.

ABSTRACT

One category of learning methods in artificial neural networks is reinforcement or graded learning. This kind of learning can be met in biological systems also. This paper presents the performance of the stochastic reinforcement learning algorithm. The performance of the algorithm with one stochastic unit and with stochastic unit and back-propagation unit is compared with the performance of the back-propagation unit that is trained using supervised learning. Although supervised learning is much faster than reinforcement learning, the latter can be used even though the desired outputs for every input are not known.

I. UVOD

Paraleleno strukturo, sestavljeno iz procesnih elementov (ti so med seboj povezani z usmerjenimi povezavami - sinapsami), ki je sposobna specifičnega porazdeljenega procesiranja informacij, imenujemo nevronska mreža. Osnovni gradniki nevronske mreže, procesni elementi, izvajajo procesiranje na osnovi svoje prenosne funkcije, trenutnih vrednosti na svojih vhodih in vrednosti v svojem lokalnem pomnilniku /1/. Pri tem igra pomembno vlogo pravilo učenja, to je pravilo, po katerem se spreminjajo vrednosti v internih pomnilnikih procesnih elementov. Nevronske mreže, ki uporabljajo eno ali več pravil učenja, morajo nujno skozi fazo učenja, ki lahko poteka v obliki :

- nadzorovanega učenja (supervised learning)
- samoorganizacije (self - organization)
- ocenjevanega učenja (graded, reinforcement learning).

Pri nadzorovanem učenju mreža potrebuje "učitelja", ki ima nalogo posredovati pravilni izhod za vsak mreži podani vhod. Mreži tako predstavimo pare (x_i, y_i), $i = 1,...,n$, pri čemer

je x_i vhod, yi pa pravilni oz. želeni odgovor na x_i.

Za samoorganizacijo je značilno, da mreža razen vhodov ne potrebuje nobene dodatne informacije, saj ji le-ti zadostujejo, da se na osnovi notranjega pravila sama ustrezno oblikuje.

V primerjavi z nadzorovanim gre pri ocenjevanem učenju za drugačno vrsto signala, ki ga mreža dobi v fazi učenja. Namesto vektorja, ki za dani vhod predstavlja želeni izhod, imamo pri ocenjevanem učenju skalarno oceno obnašanja mreže glede na neko mero. Upoštevajoč signal ocene, skuša mreža izboljšati obnašanje v smeri generiranja pravilnih izhodov. Pri ocenjevanem učenju gre tako za dvoje, za iskanje pravilnih izhodov na podane vhode in za pomnjenje pravilnih izhodov. Ker daje skalarni signal ocene pri ocenjevanem učenju manj informacije kot želeni izhod pri nadzorovanem učenju, je ocenjevano učenje običajno počasnejše od nadzorovanega, njegove prednosti pa se pokažejo v primerih, ko želeni izhodi niso vnaprej poznani, oziroma ko je kot odgovor na dani vhod možnih več alternativ (npr. pri kontroli sistemov z določeno stopnjo avtonomnosti).

II. UČENJE V BIOLOŠKIH SISTEMIH

Nevronsko računalništvo, katerega predmet zanimanja so v prvi vrsti nevronske mreže, se je razvijalo tudi pod vplivom nevrologije (znanstvene discipline, ki skuša razložiti delovanje možganov in miselnih procesov), vendar pa v zadnjem času postaja ta vpliv vzajemen. Nove arhitekture nevronskih mrež in novi koncepti ter teorije o delovanju nevronskih mrež pomagajo tudi nevrologiji na njeni poti do odkritja delovanja možganov in procesov v njih.

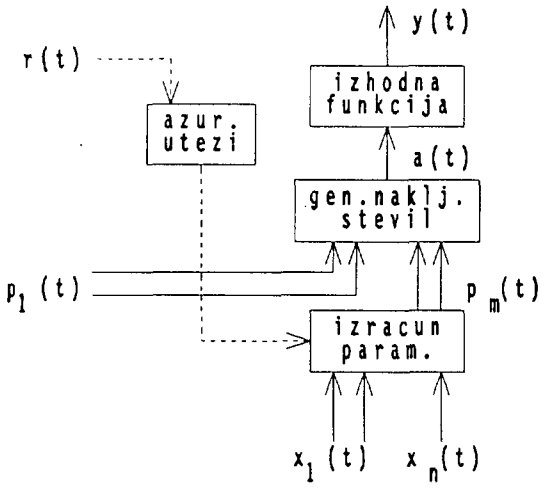
Ocenjevano učenje kot eden izmed treh možnih načinov učenja v umetnih nevronskih mrežah ima svojo "živo" vzporednico. Psihologi so v klasični teoriji učenja postavili vrsto definicij pojma "reinforcement", pri čemer pa so si te definicije v jedru enotne, da gre za operacijo okrepitve, ojačanja , utrjevanja, oziroma za sam dogodek, ki tako ojačanje oz. okrepitev povzroči. Pri tem je to, kar se ojača, običajno naučen odgovor ali pa vez med tem odgovorom in dražljajem /3/. Okrepitev je lahko pozitivna ali pa negativna. Pozitivno okrepitev predstavlja dogodek, dražljaj ali vedenje, ki, kadar je pogojen-o z odgovorom, povzroči povečanje verjetnosti za pojav tega odgovora v bodoče. Nasprotno pa negativno okrepitev predstavlja dogodek, dražljaj ali vedenje, ki, kadar je njegova ukinitev pogojena z odgovorom, poveča verjetnost tega odgovora v bodoče. Glede na to, kakšna je odvisnost okrepitve od nekega vidika odgovora (ta je lahko prostorski, časovni ali sekvenčni), so psihologi klasificirali okrepitve v tri razrede. V prvem razredu t.i. "enostavnih" okrepitev gre za samo eno vrsto pogojenosti med odgovorom in pojavom okrepitve. V drugi razred spadajo "sestavljene" okrepitve, ki so sekvenčna ali paralelna kombinacija dveh ali več "enostavnih" okrepitev. Okrepitve, ki jih ne moremo uvrstiti v nobenega od prejšnjih razredov, spadajo v razred t.i. "posebnih" okrepitev. Ocenjevano učenje (reinforcement, graded learning) morda ni najboljše poimenovanje za to vrsto učenja, vsaj v takšnem smislu ne, kot to razumejo psihologi. Glede na to, da ime dobro označi to vrsto učenja v umetnih nevronskih mrežah (ocena, kritika, ki okrepi obnašanje mreže v pravi smeri), je to poimenovanje opravičeno.

III. STOHASTIČNI ALGORITEM Z OCENJEVANIM UČENJEM

Ideja za ta algoritem, ki ga je v osnovi razvil V. Gullapalli /4/, izvira iz teorije učečih avtomatov. Stohastični učeči avtomat je

abstraktni stroj, ki akcije izbira naključno po podani verjetnostni porazdelitvi, pri tem pa iz okolja dobiva povratno informacijo, ki predstavlja ovrednotenje akcij. Upoštevajoč povratno informacijo, avtomat ažurira verjetnostno porazdelitev za akcije tako, da poveča verjetnost za pozitivno ovrednotenje akcij v bodoče.

Učeči avtomat deluje v povratni povezavi z okoljem, ki v času t pripelje na vhod avtomata vektor $\mathbf{x}(t)$. Odgovor avtomata $y(t)$ je naključno izbran glede na verjetnostno porazdelitev na intervalu $Y \subseteq R$. Signal ocene $r(t) \in R = [0,1]$, ki ga generira okolje, predstavlja oceno odgovora $y(t)$ v kontekstu vhoda $\mathbf{x}(t)$. Cilj učečega avtomata je naučiti se odgovarjati na vsak vhodni vektor $\mathbf{x}$ s takšnim odgovorom y, da bo ocena, ki jo bo za ta odgovor prejel od okolja, maksimalna. Opisani stohastični avtomat se implementira kot stohastična enota (slika 1), ki nastopa kot komponenta v mreži.



slika 1

Vhod v enoto v času t je $\mathbf{x}(t) = (x_1(t), x_2(t),...,x_n(t))$, ki se uporabi za izračun parametrov $p_1(t)$, $p_2(t)$,..., $p_m(t)$ verjetnostne porazdelitve, po kateri se naključno generira aktivnost enote. Parametre porazdelitvene funkcije lahko dobimo od zunaj ali pa so določeni kot utežena vsota vhodov, z različnim naborom uteži za vsak parameter. Aktivnost enote je tako naključna spremenljivka porazdelitve, določene s parametri $p_1(t),...,p_m(t)$. Izhod iz enote $y(t)$ je funkcija aktivnosti $y(t) = f(a(t))$, pri čemer je funkcija f (pragovna funkcija, logistična funkcija) izbrana glede na vrsto izhoda, ki ga želimo.

Za implementacijo določenega stohastičnega algoritma učenja je potrebno tako določiti :

1. porazdelitev, ki se uporabi za generiranje naključnih vrednosti aktivacije
2. funkcije, ki se uporabijo za izračun parametrov porazdelitve
3. izhodno funkcijo f
4. pravila za ažuriranje uteži

V članku je predstavljen algoritem stohastičnega ocenjevanega učenja za učenje funkcij z realnimi izhodi. Za generiranje aktivnosti enote se uporabi normalna porazdelitev $\Psi(\mu, \sigma)$. Izhod iz enote, ki je realna vrednost, je zvezna, monotona funkcija aktivnosti. Signal ocene je iz intervala $[0, 1]$. Trenutni vhodi v enoto določajo srednjo vrednost in standardno deviacijo porazdelitve, na podlagi katere se naključno generira aktivnost enote.

Učenje poteka v smislu ažuriranja parametrov normalne porazdelitve (standardne deviacije in srednje vrednosti) v smeri povečanja verjetnosti za generiranje optimalnega izhoda za vsak vhod. Pri tem je srednja vrednost μ ocena za optimalno aktivnost, standardna deviacija σ pa določa obseg iskanja okrog trenutne srednje vrednosti aktivnosti enote.

Ker naj bo srednja vrednost porazdelitve μ ocena za optimalni izhod, izračunamo μ kot uteženo vsoto vhodov :

$$\mu(t) = \sum_{i=1}^{n} w_i(t)\, x_i(t) + w_{prag}(t)$$

Za dani vhod naj bo st.deviacija σ odvisna od tega, kako blizu je trenutni pričakovani izhod optimalnemu izhodu za ta vhod. Mera za to pa je signal ocene iz okolja. Za dani vhod je tako st.deviacija σ odvisna od pričakovanega signala ocene, ki ga prav tako dobimo kot uteženo vsoto vhodov :

$$r(t) = \sum_{i=1}^{n} v_i(t)\, x_i(t) + v_{prag}(t)$$

Pričakovani signal ocene uporabimo za izračun st.deviacije :

$$\sigma(t) = s(r(t)),$$

kjer je funkcija s monotono padajoča nenegativna funkcija pričakovanega signala ocene.

Na osnovi $\sigma(t)$ in $\mu(t)$ se izračuna aktivnost a(t) kot naključna spremenljivka normalne porazdelitve:

$$a(t) \sim \Psi(\mu(t), \sigma(t)) .$$

Izhodna funkcija f preslika aktivnost enote a(t) v izhod enote y(t) :

$$y(t) = f(a(t)), \text{ pri čemer je } f(x) = \frac{1}{1 + e^{-x}} .$$

Pravila za ažuriranje uteži, ki določajo srednjo vrednost:

$$w_i(t+1) = w_i(t) + \alpha\, \Delta_w(t)\, x_i(t)$$

$$w_{prag}(t+1) = w_{prag}(t) + \alpha\, \Delta_w(t)$$

$$\Delta_w(t) = \Big(r(t) - r(t)\Big) \left(\frac{a(t) - \mu(t)}{\sigma(t)}\right)$$

Pravila za ažuriranje uteži, ki določajo st.deviacijo :

$$v_i(t+1) = v_i(t) + \beta\, \Delta_v(t)\, x_i(t)$$

$$v_{prag}(t+1) = v_{prag}(t) + \beta\, \Delta_v(t)$$

$$\Delta_v(t) = r(t) - r(t)$$

Pri tem sta α in β parametra učenja.

Osnovni cikel učenja poteka po naslednjih korakih :

1. okolje poda enoti vhodni vektor $x_i(t)$, $i = 1,...,n$

2. enota uporabi vhodni vektor za izračun srednje vrednosti $\mu(t)$ (uporabijo se uteži $w_i(t)$ in w_{prag})

3. enota izračuna pričakovani signal kritike r(t), ki ga uporabi za izračun $\sigma(t)$ (pri tem se uporabijo uteži $v_i(t)$ in v_{prag})

4. enota izračuna aktivnost $a(t) \sim \Psi(\mu(t), \sigma(t))$ in izhod $y(t) = f(a(t))$

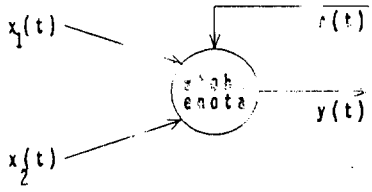
5. okolje ovrednoti izhod in odpošlje signal ocene r(t)

6. r(t) omogoči ažuriranje uteži

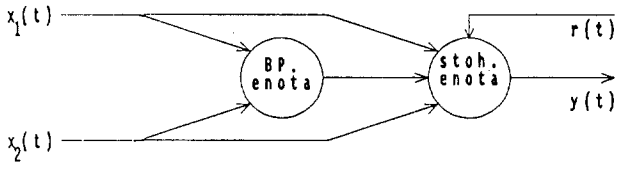
IV. OBNAŠANJE ALGORITMA

Delovanje algoritma je preizkušeno na več opravilih, ki so definirana kot množica parov "dražljaj - odgovor" iz $[0, 1]^2 \times (0, 1)$. Za vsa opravila se je uporabila najprej samo ena stohastična enota (slika 2), nato stohastična enota v povezavi z back-propagation enoto (slika 3) in primerjalno še ena sama back-propagation enota (učenje te enote je potekalo v obliki nadzorovanega učenja s

parametrom učenja 0.1). Hitrost konvergence za vse tri primere je razvidna iz grafov (slike 4, 5 in 6).



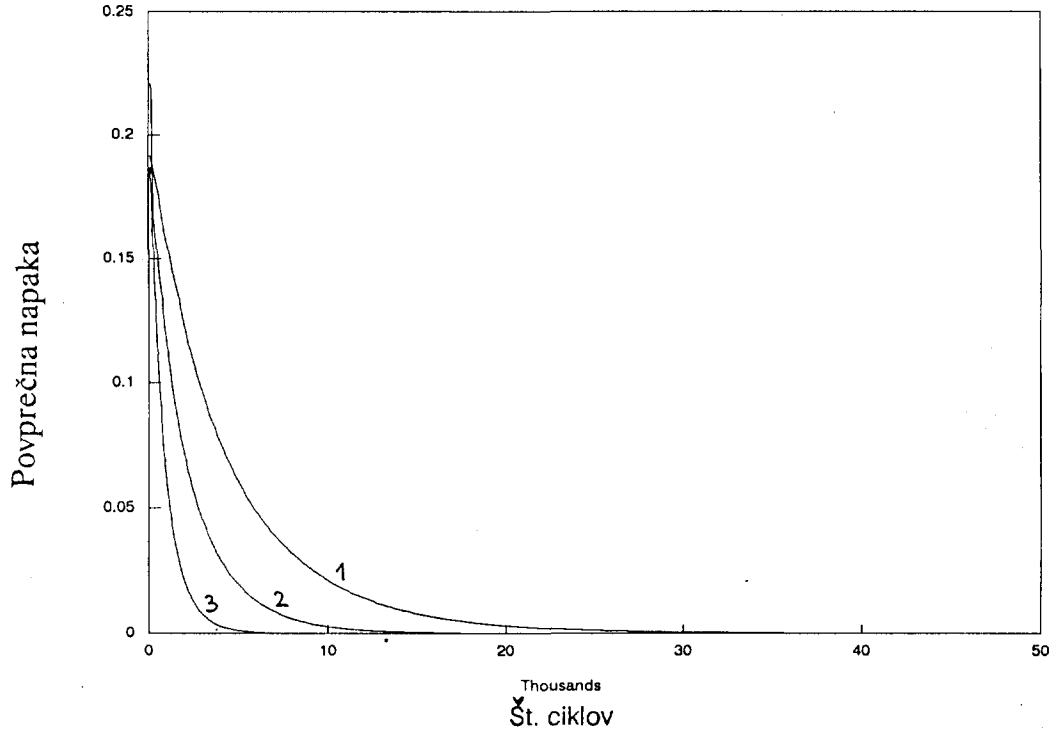
slika 2



slika 3

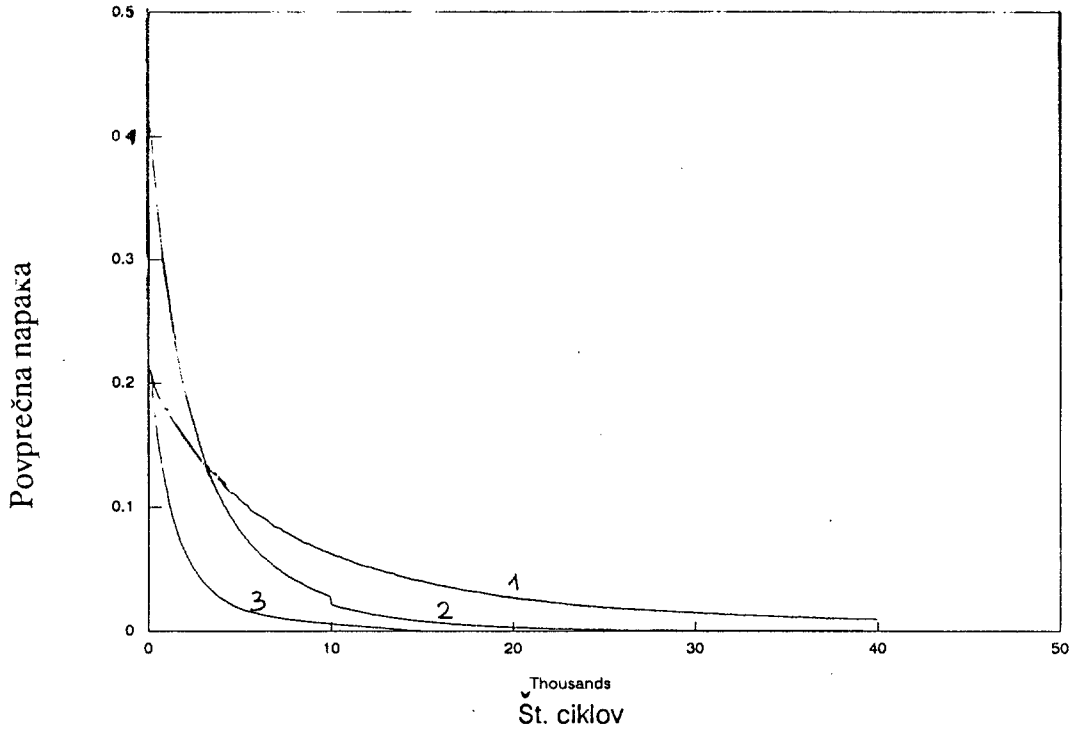
Delovanja algoritma v primerih ene stohastične enote

Ob začetku učenja so vse uteži enote postavljene na 0, vrednosti parametrov učenja pa so $\alpha = 0.5$ in $\beta = 0.7$. Na vhod enote pripeljemo v vsaki fazi učenja vse vhodne vektorje v naključnem vrstnem redu, pri tem pa se osnovni cikel učenja ponovi za vsak vhodni vektor. Ob koncu točke 4 osnovnega cikla učenja se izračuna napaka kot razlika med dejanskim in želenim izhodom, ki se uporabi za določitev signala ocene. Sledi ažuriranje uteži. Učenje poteka tako dolgo, dokler povprečna napaka ne pade pod določeno mejo, oziroma po preteku določenega števila ciklov učenja.



Krivulja št. 1 prikazuje napako za stoh. enoto. Krivulja št. 2 prikazuje napako BP in stoh. enote in krivulja 3 prikazuje napako ene same BP enote. V vseh primerih smo učili z vektorji : (0.2, 0.6) -> 0.3 in (0.8, 0.3) -> 0.7

Slika 4



Krivulja št. 1 prikazuje napako za stoh. enoto. Krivulja št. 2 prikazuje napako BP in stoh. enote in krivulja 3 prikazuje napako ene same BP enote. V vseh primerih smo učili z vektorji : (0.4, 0.7) -> 0.3, (0.1, 0.3) -> 0.6 in (0.6, 0.2) -> 0.9.

Slika 5

Signal ocene iz okolja je določen kot :

$r(t) = 1 - |\text{napaka}|$,

pri tem je $|\text{napaka}| <= 1$.

Delovanja algoritma v primerih stohastične in back-propagation enote

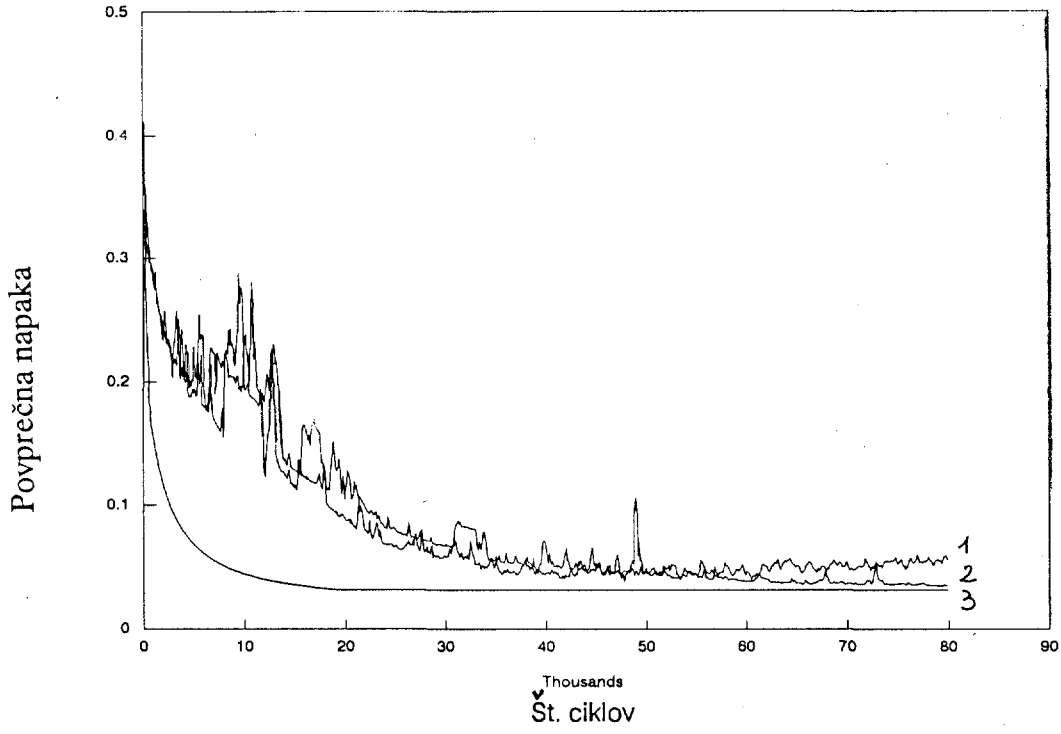
Hibridna mreža, sestavljena iz stohastične izhodne enot in enote v skritem nivoju, ki je tipa back-propagation, izkazuje boljše obnašanje kot ena sama stohastična enota.

V taki mreži se kot aproksimacija napake na izhodu uporabi vrednost $\Delta_w(t)$. Poleg tega se osnovnemu ciklu učenja doda še korak, kjer se izraz $\Delta_w(t)$ iz izhodne enote propagira nazaj kot napaka na skrito enoto in s tem omogoča ažuriranje uteži back-propagation enote.

Ob začetku učenja so uteži za back-propagation enoto postavljene naključno, za stohastično enoto pa so 0, parameter učenja za stohastično enoto $\beta = 0.7$, parameter α pa je postavljen različno za povezave na vhode $\alpha = 0.5$ in za povezave na skrito enoto $\alpha = 0.1$. Parameter učenja za back-propagation enoto je 0.1. V vsaki fazi učenja se na mrežo pripeljejo vsi vhodni vektorji, za vsakega se ponovi osnovni cikel učenja. Signal ocene je določen enako kot v primeru ene stohastične enote.

V. ZAKLJUČEK

Rezultati so pokazali, da stohastična enota, povezana z back-propagation enoto izkazuje boljše obnašanje kot ena sama stohastična enota. Ta razlika je sicer v nekaterih primerih (slika 6) majhna.



Krivulja št. 1 prikazuje napako za stoh. enoto. Krivulja št. 2 prikazuje napako BP in stoh. enotė in krivulja 3 prikazuje napako ene same BP enote. V vseh primerih smo učili z vektorji : (0.1, 0.1) -> 0.1, (0.1, 0.9) -> 0.1, (0.9, 0.1) -> 0.1 in (0.9, 0.9) -> 0.9.

Slika 6

Konvergenca učenja v primeru ene same stohastične enote in stohastične enote, povezane z back-propagation enoto, je bistveno počasnejša od konvergence učenja pri back-propagation enoti. Faktor, ki vpliva na obnašanje vseh sistemov, ki se učijo z ocenjevanim učenjem, je kvaliteta signala ocene iz okolja. Zato lahko včasih izboljšamo obnašanje mreže, če v signal ocene vgradimo več informacije, ki je specifična za neko opravilo.

Literatura:

/1/ Robert Hecht-Nielsen, Neurocomputing, Addison-Wesley,1990

/2/ Hinton, Connectionist Learning Procedures,Artificial Intelligence 40, 1989

/3/ Dictionary of psychology,

/4/ V.Gullapalli, A Stochastic Reinforcement Learning Algorithm for Learning Real-Valued Functions, Neural Networks, Vol.3,1990

/5/ R.A. Leaver, P. Mars, Stochastic Computing and reinforcement neural networks, Conference on Artificial Neural Networks, London, 1989

DIGITALIZACIJA IN VEKTORIZACIJA ČRTNIH RISB VELIKIH DIMENZIJ

INFORMATICA 4/91

A. Dedić
R. Murn
D. Peček
INSTITUT JOŽEF STEFAN
LJUBLJANA
Odsek za računalništvo in informatiko

Keywords: image analysis and processing, pattern recognition, vectorization, GIS

Programska orodja in aplikacije, ki tečejo na sodobnih računalniških sistemih omogočajo zajem, uporabo in upravljanje prostorsko porazdeljenih podatkov. Zaradi velike količine podatkov, mora biti vnos hiter in učinkovit. Avtomatski vnos grafičnih podatkov nam omogočajo postopki za natančno digitalizacijo in vektorizacijo grafičnih predlog, ki so narisane na papirju in njemu sorodnih medijih.V delu so proučeni postopki za digitalizacijo in vektorizacijo črtnih risb velikih dimenzij. Na osnovi ovrednotenja teh postopkov in zahtev uporabnikov so izdelani osnovni pogoji za kakovostno in hitro digitalizacijo grafičnih predlog in transformacijo rasterske slike v vektorski zapis. To so: natančnost postopka, uporabnost rezultata in hitrost. Izdelan je lasten postopek za digitalizacijo in vektorizacijo, ki optimalno izpolnjuje navedene pogoje.

DIGITALIZATION AND VECTORIZATION OF LARGE SCALE DRAWINGS AND MAPS

Modern programing tools and applications running on up-to-date computer systems enable efficient managing of spatial distributed data. Due to enormous amount of image data, graphic data input must be extremely fast and reliable. In the paper raster to vector conversion techniques are discussed. There are two basic requests: usefulness of the result, and the speed of data acquisition. Dedicated raster to vector converter is shown.

Uvod

Človeški rod je že od pradavnine iz različnih razlogov zbiral podatke o svojem okolju. Težišče tega zbiranja so bili v glavnem geografski in topografski podatki. V 19. stoletju so bila prvič izvedena triangulacijska dela kot osnova novih kartografskih projektov. Prav tako se začne razvijati sistem zemljiškega katastra. V našem stoletju je stekel razvoj še mnogo hitreje.

Nasprotno od hitrega razvoja tehnik pridobivanja podatkov o prostoru je tehnika hranjenja teh podatkov napredovala le počasi. Od kamnitih plošč, papirusa je prišel papir, ki se je obdržal do današnjih dni. V zadnjih dveh dekadah se za shranjevanje in obdelavo podatkov nezadržno širi uporaba računalniške tehnologije. Razviti so računalniški sistemi, ki zagotavljajo učinkovito zbiranje, vodenje, povezovanje in vrednotenje geometrijskih podatkov o okolju. Take sisteme imenujemo Geografski Informacijski Sistemi (*GIS*).

Zajemanje podatkov je časovno in finančno najzahtevnejša komponenta *GIS*-a. Po nekaterih podatkih je delež vnosa podatkov *80* odstotkov celotne investicije. Poleg vseh ostalih zahtev zajemanja, kot sta hitrost in enostavnost, je bistveni pogoj pri zajemanju ohranjanje izvorne natančnosti podatkov. Sodobna tehnologija omogoča relativno enostavno in hitro zajemanje analognih grafičnih predlog (kart, map) z pomočjo posebnih linijskih kamer, in pretvorbo v digitalno obliko. Rezultat se shrani v računalnik v rasterski obliki. Članek obravnava pretvorbo rasterskih slik v vektorsko obliko.

Definicija problema

Digitalno sliko, lahko v rasterskem zapisu enostavno in hitro prikažemo na računalniškem zaslonu. Vendar ta oblika zapisa ne nosi informacije o vsebini grafične predloge.

Vektorski zapis (vektorska slika) grafičnih podatkov je primerna za računalniško obdelavo in nosi strnjen opis vseh informacij o grafični predlogi, ki jo ponazarja. Zato je

potrebno razviti ustrezen računalniški postopek za transformacijo rasterskih podatkov v vektorske. Transformacija mora izpolnjevati nekatere zahteve. Najvažnejše je, da se ohrani izvorna natančnosti podatkov, saj je njihova vsebina informacija o metriki prostora, ki je zakonsko predpisana.

Računalniški postopek mora omogočati enostaven, hiter in natančen vnos podatkov iz grafičnih predlog v računalniško podprt geografski informacijski sistem [1,2,3].

Celotni postopek je razdeljen na dva zaporedna, smisleno in programsko ločena dela:

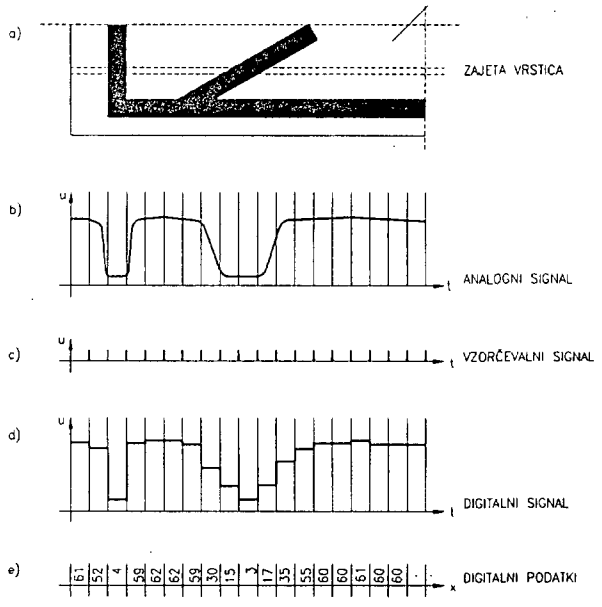
a) digitalizacija črtnih risb velikih dimenzij,

b) vektorizacija dobljenih podatkov.

Prvi del vsebuje opis problematike digitalizacije grafičnih predlog velikih dimenzij. V drugem delu so detaljno analizirane zahteve za transformacije podatkov iz rasterskega v vektorski zapis.

Digitalizacija črtnih risb

Pod pojmom "*digitalizacija*" razumemo proces zajemanja črtnih risb z ustrezno kamero (linijsko - enodimenzijsko ali ploskovno - dvodimenzijsko) in transformacijo analognih



Slika 1. Analogni signal in digitalni podatki

signalov (podatkov) v digitalno obliko. Taki transformirani digitalni podatki so primerni za obdelavo in za hranjenje v digitalnem računalniku. Transformacija zveznih analognih signalov v analogne vzorce se izvaja z elektronskim vezjem v kameri. Analogne signale vzorčimo v predpisani frekvenci in določamo digitalno vrednost vzorcev v odvisnosti od amplitude *(Slika 1)*. Število digitalnih razredov je ponavadi iN-ta potenca števila 2. Značilne vrednosti za *N* so *1, 4, 6, 8, ...* . Najvišji amplitudi odgovarja najvišji digitalni razred in najnižji amplitudi, najnižji razred. Vse vmesne amplitude se zaobkrožijo navzdol na ustrezni razred.
Za črtne risbe večjih dimenzij (standardni formati *A2, A1, A0* in večji) je značilno, da z današnjo tehnologijo ni možna enostavna in cenovno dostopna digitalizacija z dvodimenzijsko kamero zaradi premajhne ločljivosti ustreznih senzorjev. Za te namene se je uveljavila uporaba ustrezno prirejenih linijskih kamer. Na tržišču je uveljavljen izraz "*SKENER*"; imenovali ga bomo "*linijski digitalizator*". Z linijskim digitalizatorjem lahko digitaliziramo grafične predloge, ki so omejene z fizično širino linijskega senzorja (ponavadi več kot en meter) in teoretično neomejene dolžine. Dolžina je omejena z velikostjo pomnilnika v računalniku v katerega se zapisujejo podatki iz digitalizatorja. Važen podatek za linijski digitalizator je ločljivost zajemanja. Standardne ločljivosti so *100, 200, 300* in *400* točk na inčo *(dots per inch - DPI)*. Novejši digitalizatorji dosežejo do *600 DPI.*

Za potrebe aplikacij katerih cilj je zapis segmentov iz grafične predloge v matematični obliki *(vektorizacija)*, je najbolj ugodno, če je slika zapisana v dveh nivojih: belo za ozadje slike in črno za črte na sliki. Črtne risbe, ki so najbolj zanimive in so vplivale na razvoj magistrske naloge so:

- *katasterske mape standardnih meril,*
- *geografske mape različnih meril,*
- *geodetske, gozdarske, elektrovodne, komunalne, in druge mape,*
- *inženirski načrti vseh vrst in*
- *druge grafične predloge, ki so izrisane z črtami kakršne koli oblike.*

Črtne risbe so največkrat risane v eni barvi na kontrastni podlagi. Obstajajo tudi take, ki so risane v več barvah, vendar na različnih slojih - prozornih folijah, kar obravnavamo kot več različnih, enobarvnih predlog. Za starejše tipe grafičnih predlog, ki so risane v več barvah na eni sami podlagi lahko uporabimo barvne filtre pri digitalizaciji *(barvna separacija)*, tako da dobimo več različnih enobarvnih predlog. To je značilno za veljavne katasterske načrte iz obdobja Marije Terezije. Takšni načrti še zmeraj pokrivajo približno *80* procentov površine Republike Slovenije. Osnovni problem pred vsebinsko obdelavo digitalizirane slike je pravilen izbor algoritmov (izbor uveljavljenih in/ali izdelava novih). Kompleksno sliko je namreč potrebno razbiti na eno ali več homogenih površin, ki opisujejo posamezne logične nivoje. Ta problem imenujemo segmentizacije ali razčlenjevanje slike. Popularna metoda segmentizacije slike je "rezanje" slike na enem sivinskem nivoju *(image thresholding)*. Ta metoda je računsko bolj enostavna in hitrejša kot druge metode za segmentizacijo (odkrivanje robov na sivinski sliki z različnimi operatorji ali razne tehnike, ki slonijo na odkrivanju homogenih področij na sliki).

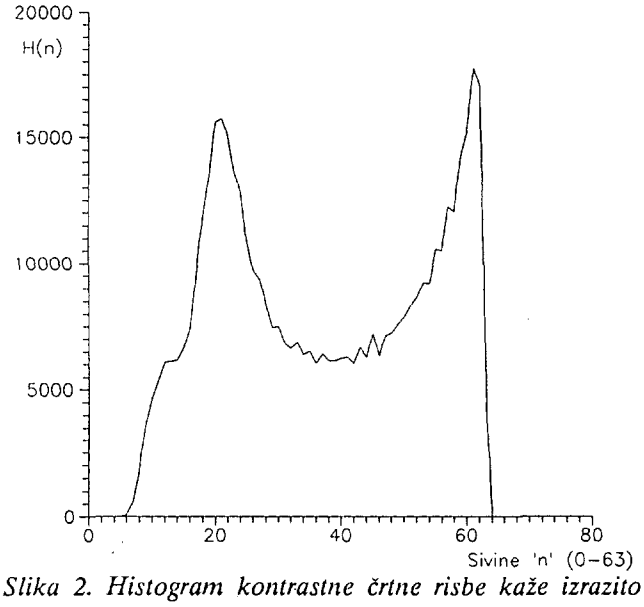
Metoda rezanja slike je v bistvu metoda iskanja optimalne *PRAGOVNE* vrednosti. Tako na enostaven način dobimo uporabno dvonivojsko sliko. V splošnem primeru je ta metoda dosti bolj zahtevna, če so črte na sliki, ki jih je treba segmentizirati v različnih sivinah (npr. *N* sivin) in je zato potrebno poiskati več ustreznih pragovnih vrednosti *(N - 1)*.

Za segmentizacijo kakovostnih črtnih risb je zadostna in potrebna predstavitev z dvemi nivoji, *črno - belo.* V teh primerih kaže tudi histogram sivinskih vrednosti izrazito dvopasovno obliko: pas, ki odgovarja točkam na črtah in pas, ki odgovarja točkam ozadja *(Slika 2).* Na tak način lahko sliko *f* segmentiziramo na dva dela tako, da upoštevamo razmerje sivinskih vrednosti *G(x,y)* proti eni sivini *T.* Vrednost *T* - ja leži med maksimumom sivin črt in maksimumom sivin ozadja.

Ta enostavni kriterij za vse točke slike *f* se da predstaviti kot:

- *če je G(x,y) > = T potem je točka iz skupine črt ali skupine 1,*
- *sicer je točka iz skupine točk ozadja ali skupine 2.*

Vrednost *T* imenujemo pragovna vrednost *(threshold)*, postopek pretvorbe slike iz večsivinskega zapisa v dvonivojski pa binarizacija.



Slika 2. Histogram kontrastne črtne risbe kaže izrazito dvopasovnost

Digitalizacija in binarizacija pripravi sliko za nadaljno obdelavo, v našem primeru vektorizacijo. Kakovost digitalizacije in binarizacije vpliva na rezultat vektorizacije črtne risbe (ne glede na uporabljeno metodo vektorizacije) [4,5].

Linijski digitalizator omogoča nastavitev pragovne vrednosti za binarizacijo grafične predloge. Kakovost tako dobljenih binarnih slik je zadovoljiva samo za del novejših črtnih risb, ki so risane na posebnih predlogah s stabilnimi, kvalitetnimi peresi. Dobljeni rezultati za ostale črtne risbe so slabši. Razlog je pragovna vrednost, ki je nastavljena globalno, enotno za celo grafično predlogo, ne glede na spreminjanje odtenkov sivine narisanih črt po celi sliki in spreminjanje odtenkov barve papirja kot ozadja [6].

Na slabšanje rezultatov vplivajo:

- *slaba kvaliteta papirja, ki s staranjem spreminja barvo,*
- *če je papir izrabljen, izdrsan (izdrgnjen), umazan na kakršen koli način, lepljen z neustreznimi lepili (selotejp trakovi),*
- *če je risan z slabimi peresi z nestabilnim črnilom, ki s časom spreminja barvo, ali s preveč vodenim črnilom.*

Te slabosti se kažejo na digitalni, globalno binarizirani sliki kot:

- *črte so neenakomerno debele, na enem mestu pretanke, drugje predebele,*
- *prekinjene črte ali jih sploh ni,*
- *na mestih, kjer je gostota črt zelo velika (naselja, križišča), so črte odebeljene, ponekod zlite v črne ploskve.*

Vse metode iskanja pragovne vrednosti za digitalno, večsivinsko sliko lahko v splošnem razdelimo na dve skupini:

- *metode iskanja globalne pragovne vrednosti (global threshold selection) in*
- *metode iskanja spremenljive (lokalne, dinamične, spremenljive) pragovne vrednosti (variable threshold selection).*

S prvimi določimo za celo sliko enotno pragovno vrednost. Ostale metode določajo pravila, po katerih se pragovna vrednost spreminja po sliki od točke do točke.

Lastna metoda (variabilno načelo [7])

Pragovna vrednost T_n se za vsako točko *n* računa kot:

P_2	P_3	P_4
P_1	P_0	P_5
P_8	P_7	P_6

Slika 3. Oznake indeksov sosednjih točk

$$T_n = [\, C_n \sigma_n + \mu_n] \;, \qquad n = 1, 2, 3, \ldots, N$$

N je število vseh točk na sliki f, μ_n je srednja vrednost tekoče točke in njenih osmih sosedov *(Slika 3)*.

$$\mu_n = \sum_i P_i \;, \qquad (i = 0, 1, \ldots, 8)$$

σ_n je standardno odstopanje devetih točk od srednje vrednosti μ_n:

$$\sigma_n = [\, \sum_i \frac{(P_i - \mu_n)^2}{9}]^{(1/2)} \;, \quad (i = 0, 1, \ldots, 8)$$

Koeficijentu C_n smo dali ime *"pragovni koeficijent"* in je odvisen od histograma cele slike $H(n)$:

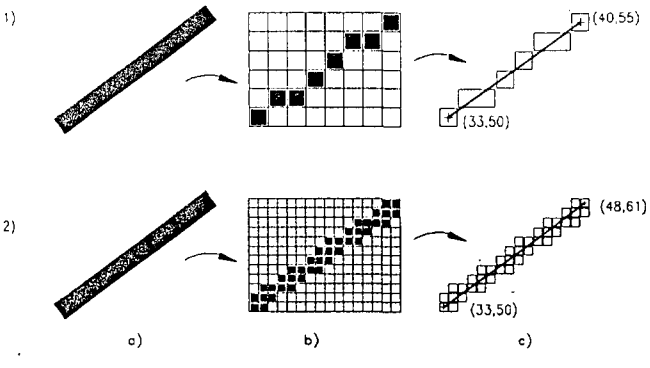
$$C_n = \frac{k\,[H(n)]}{N}$$

Koeficijentu C_n smo dali ime *"pragovni koeficijent"* in je odvisen od histograma cele slike $H(n)$:

$H(n)$ je število točk, ki imajo sivinsko vrednost n na celi sliki f. Vrednost k je konstanta. Dodali smo jo za izboljšanje rezultatov in je odvisna od kvalitete načrta. Konstanta k se določa z izkušnjo in ima vrednost med *1* za boljše predloge in *0.65* za slabše.

Vektorizacija črtnih risb

Črtna risba, ki je digitalizirana in binarizirana, je pripravljena za glavni del obdelave, vektorizacijo. Pod pojmom *"vektorizacija"* razumemo transformacijo zapisa črte iz rasterskega v vektorski zapis in jo imenujemo tudi *"raster-vektor"* transformacija. Rasterski zapis črt je digitalizirana in binarizirana slika teh črt *(Slika 4)*. Omejimo se samo na pravokotno rastersko mrežo, ki je zgrajena iz kvadratnih rasterskih celic (obstoja tudi heksagonalna, itd).



Slika 4. Digitalizirana črta v dveh ločljivostih 1 in 2:
(a) Narisana črta na papirju,
(b) rasterska slika črte v različnih ločljivostih,
(c) vektorski zapis črte .

V našem primeru vektorski zapis definiramo kot matematičen opis črte z dvema končnima točkama, koordinatami v dvodimenzijskem, x - y prostoru. Z inverzno preslikavo *"vektor - raster"* se pogosto srečujemo v računalniškem okolju, ko je potrebno črte, ki so zapisani v vektorski obliki, izrisati na neko rastersko podlago (računalniški monitorji, matrični tiskalniki itn). Matematične črte nimajo debeline, pri rasterski predstavitvi pa jim jo moramo prirediti - najmanj eno slikovno celico.

Idealno rastersko sliko neke črtne risbe dobimo, če so znani vsi vektorji na tej risbi in jih izrišemo na rasterski podlagi z omenjenimi algoritmi. Preslikava nazaj, iz idealne rasterske slike v vektorsko, je enolična samo, če je znan algoritem po katerem je potekala preslikava v prvi smeri. Razlog je v tem, da lahko skozi rastersko črto, debelo eno ali več celic, povlečemo neskončno mnogo vektorjev.

Digitalizirana in binarizirana slika črtne risbe je rasterska slika, ki ni idealna. Položaj vektorjev na risbi seveda ni znan, saj je bistvo vektorizacije prav odkrivanje le-teh. Torej, ta preslikava ni enolična. Iz ene rasterske slike lahko dobimo neskončno mnogo različnih, medseboj podobnih vektorskih slik. Da bi to število zmanjšali, oziroma preslikavo čim bolj poenotili, lahko postavimo različne zahteve, ki jih morajo izpolnjevati dobljene vektorske slike.

Večina črtnih risb, ki jih obdelujemo, vsebuje metriko o prostoru, ki ga predstavljajo. Iformacije o tem prostoru (ali večina le-teh) so v narisanih črtah na papirju. To pomeni, da bo vsaka, še tako majhna napaka pri transformaciji pomenila popačenje v novi vektorski predstavitvi prostora. Primer tega je katasterski načrt, na katerem so narisane zemljiščne parcele. Davek na zemljo je odvisen od površine parcele. Površina večjih parcel se vidno spreminja pri še tako majhnem premiku dolgih mej - črt.

Velja da originalna črtna risba ne vsebuje idealne metrike o prostoru temveč samo približek. Zato je splošna zahteva pri vektorizaciji, da se dobljena vektorska slika čim bolj ujema z rastersko.

Kakovost vektorizacije je odvisna od uporabljene metode. Preučili smo in testiral vse dostopne metode iz literature [8–11] in kot rezultat tega in zahtev uporabnikov postavili lastne zahteve za pogoje vektorizacije. Na osnovi teh zahtev in upoštevajoč delne rešitve iz omenjene literature, smo izdelal lastno metodo za vektorizacijo. V nadaljevanju najprej sledi pregled nekaterih bolj zanimivih in pogostejše uporabljenih metod vektorizacije.

Pregled postopkov

Velika večina postopkov vektorizacije črtnih risb uporablja kot prvi korak (ali predobdelavo) različne metode za tanjšanje črt na rasterski sliki na debelino ene rasterske celice *(Tabela 1)*. Razlog za tako široko uporabo te predobdelave je v tem, da so postopki za nadaljno obdelavo teh črt, debelih samo eno celico, že razviti in obdelani znotraj postopkov, ki so namenjeni za druge različne potrebe.

Večinoma so to različne metode iskanja in izločevanja črt in črtnih oblik kot so:

- *operatorji za iskanje robov objektov na sliki,*
- *konturni filtri,*
- *sledenje roba objekta na sliki (border tracking),*
- *algoritmi za odkrivanje linijskih segmentov na sliki itn.*

Zato bi lahko v splošnem vse postopke za vektorizacijo razvrstili v dve skupini:

- *skupina A: tiste, ki kot prvi korak pri obdelavi uporabljajo različne metode za tanjšanje rasterskih črt*
- *skupina B : tiste, ki slonijo na vektorizaciji rasterskih črt originalne širine.*

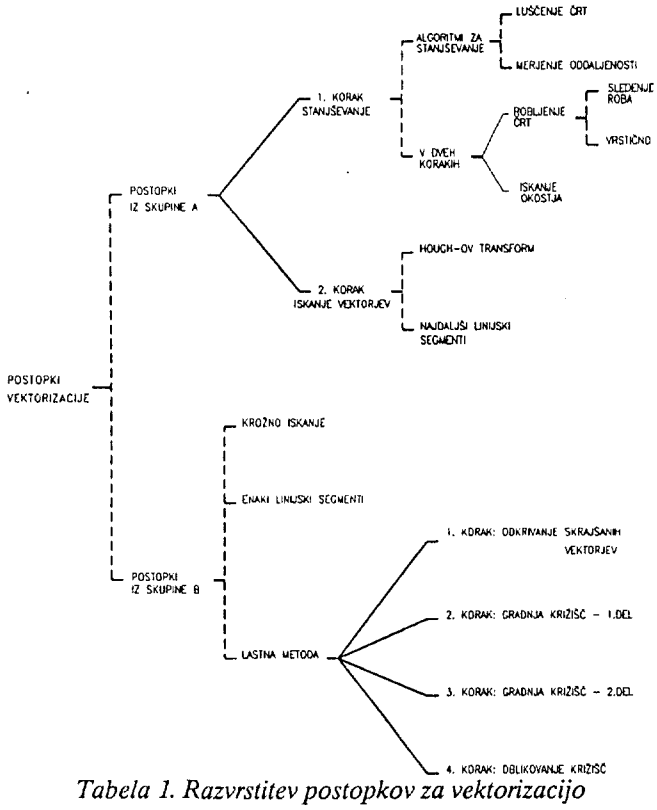


Tabela 1. Razvrstitev postopkov za vektorizacijo

Merila za vrednotenje postopkov za vektorizacijo

Da bi lahko ovrednotili postopke vektorizacije, moramo najprej določiti merila. Na prvem mestu so to zahteve ljudi iz stroke, ki bodo uporabljali programske rešitve za vektorizacije.

Za vrednotenje smo izbrali tri osnovna merila:

1. natančnost metode,

2. uporabnost dobljenega rezultata,

3. časovna zahtevnost postopka.

Po prvem merilu vrednotimo natančnost dobljenih rezultatov (ujemanje vektorskih črt z rastersko sliko). Minimalna zahtevana natančnost je, da vektor leži znotraj točk črte na rasterski sliki.

Drugo merilo so zahteve in želje uporabnikov. Naše izkušnje so pokazale, da zaenkrat ni mogoče implementirati univerzalnega algoritma, katerega rezultat bi bil sprejemljiv za različne vrste uporabnikov. To pomeni, da je potrebno vektorizacijske algoritme prilagoditi vsakemu uporabniku posebej. To se največkrat da doseči z nastavljanjem različnih parametrov vektorizacijskega procesa.

Pomembno je tudi tretje merilo. Zaradi izredno velikega števila risb, ki jih je treba obdelati, morajo biti postopki prilagojeni. Posebej še, če naj ti algoritmu tečejo na cenenih računalnikih tipa *IBM PC*. Izpolnjevanje vseh treh pogojev je potreben in zadosten pogoj za kakovostno vektorizacijo.

Lastni postopek za vektorizacijo

Osnovno vodilo in smernice pri izdelavi lastnega sistema za vektorizacijo so bili trije osnovni pogoji. Prva ugotovitev je bila, da je potrebno celotni postopek razbiti v več enostavnejših korakov. Iskanje postopkov s katerimi bi v enem prehodu dobili uporaben končen rezultat ne daje zahtevanih rezultatov.

1. KORAK: Odkrivanje skrajšanih vektorjev

Osnovni problem vektorizacije je ogromna količina podatkov, ki jih je treba obdelati. Na primer: za črtno risbo dimenzije *800 x 700 mm* (katasterski načrt), ki je digitalizirana,z ločljivostjo *600 DPI* dobimo *320* milijonov črno belih točk ali 40 *Mby* (Mega zlogov) podatkov. Z zapisom podatkov v kodah tekočih dolžin dosežemo faktor kompresije približno *5*, to je *8 Mby*. To je še vedno veliko, vendar bistveno lažje za obdelavo.

Postopek vektorizacije mora zaradi ogromne količine podatkov teči na komprimirani sliki. Zapis v obliki kod tekočih dolžin je zelo ugoden, ker je komprimiran in ima obliko iz katere se hitro ugotovi potek rasterskih črt. Zato je tak zapis rasterske slike osnova za lasten postopek vektorizacije.

Obdelava rasterske slike mora teči zaporedno, od enega konca k drugemu. To je v skladu z zapisom slike. Kakršna koli obdelava v naključni smeri (recimo: sledenje rasterski črti po sliki) je časovno zelo zahtevna. Zato smo za prvi korak izbrali obdelavo slike v eni smeri: od zgoraj navzdol (vrstico za vrstico). S tako obdelavo občutno zmanjšamo število posegov računalnika do zunanjih enot na katerih je slika zapisana *(trdi disk)*. Za zapis rasterske slike v kodah tekočih dolžin je karakteristično, da so za vsako vrstico zapisane dolžine nizov izmenično črnih ali belih točk. Tem podatkom rečemo *črna* ali *bela polja*.

Ko pri obdelavi slike naletimo na prvo vrstico, ki ni prazna, vpišemo v posebna polja podatkov, vsako črno polje posebej. V teh poljih vodimo evidenco o začetih črtah *(Slika 5.A)*. Za vsako naslednjo vrstico navzdol (tekoča vrstica) testiramo vsako začeto črto posebej, če se nadaljuje z enim izmed črnih polj v tej vrstici.

Če se črta nadaljuje, potem testiramo naslednje pogoje po vrsti (če je eden izmed pogojev izpolnjen, ne testiramo naprej):

a) če je črno polje s katerim se črta nadaljuje tudi nadaljevanje za eno ali več sosednjih začetih črtah *(Slika 5.D)*,

b) če v tekoči vrstici obstoja še eno ali več črnih polj, ki so nadaljevanje testirane začete vrstice *(Slika 5.C)*,

Če so testi *a)* in *b)* negativni potem:

c) povlečemo ravno črto od sredine prvega črnega polja testirane črte do sredine zadnjega črnega polja (za katero je črta podaljšana). Testiramo, če eno izmed črnih polj vmes odstopa od narisane črte za več, kot je predpisano odstopanje r_0. Obstajata dve možnosti:

c_1) če je narisana črta znotraj vseh prejšnjih črnih polj, potem v polje podatkovo tekoči črti dopišemo zadnje črno polje *(Slika 5.E)*,

c_2) če celotna črta ni znotraj vseh črnih polj, potem začeto črto končamo s črnim poljem v prejšnji vrstici in začnemo novo s črnim poljem v tekoči vrstici *(Slika 5.F)*.

Če je pogoj *a)* pozitiven (to je slučaj, če se testirana črta deli v eno ali več novih črt), potem se tekoča črta konča (kot črta od njenega začetka do črnega polja prejšnje vrstice) in se začne evidenca o eni ali več črtah z začetkom v tekoči vrstici.

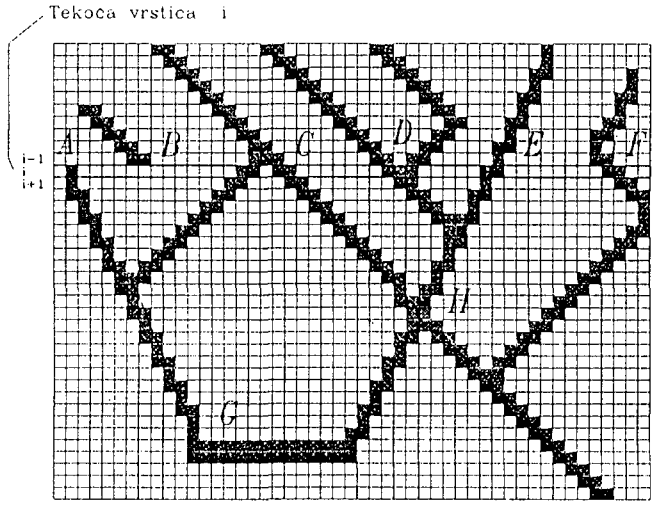
V primeru *b)* (slučaj, da se ena ali več prej začetih črt združuje v eno novo) se v prejšnji vrstici konča ena ali več začetih črt in se začne evidenca o novi črti z začetkom v tekoči vrstici.

Če se ena izmed črt ne nadaljuje v tekoči vrstici (niso izpolnjeni pogoji za nadaljevanje), se le-ta konča v prejšnji vrstici *(Slika 5.B)*. Polja podatkov v katerih smo vodili evidenco o končanih črtah sproti sproščamo.

Koordinate vseh končanih črt *(vektorje)* shranimo na posebno datoteko. Pri shranjevanju se lahko rešimo nepotrebnih kratkih vektorjev, ki večinoma predstavljajo vektorizirane napise in simbole na grafičnih predlogah. Če je vektor krajši od najkrajše vnaprej določene dolžine r_1, potem ga ne shranimo *(Slika 5.H)*.

V vsakem križišču po prvem koraku, ostanejo luknje *(Slika 5)*. V prvem koraku križišč ne gradimo. To je v bistvu osnovna ideja, na kateri je bil zasnovan celoten postopek:

Za uspešno vektorizacijo črtnih risb, ki vsebujejo metriko o prostoru je ključnega pomena natančno identificirati vektorje narisanih črt, oziroma koordinate dveh končnih točk vsake posamezne črte (vsaka črta je enolično definirana s svojima končnima točkama). Če je črta samostojna (prosta na obeh koncih), potem končne točke ni težko določiti. Problem je, da so večinoma ti dve točki v vozliščih treh ali večih črt (črte se združujejo ali razdružujejo v vozliščih). Zelo težko je natančno določiti koordinato posameznega vozlišča oziroma eno končno točko vseh črt, ki so združene v vozlišču. Na osnovi tega smo prišli do večkoračne rešitve: v prvem koraku vse črte skrajšamo za del, ki leži v vozlišču, (črte vektoriziramo samo do vozlišča). S postopki v naslednjih korakih še oblikujemo vsa vozlišča.



Slika 5. Prvi korak vektorizacije

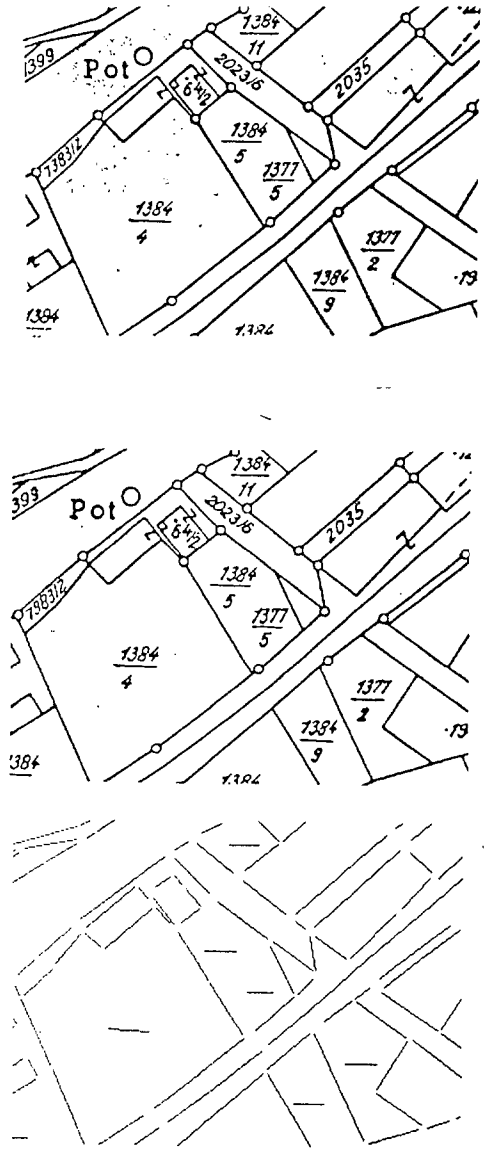
S testiranjem navedenih pogojev v prvem koraku: *a)*, *b)*, in *c)* hitro in enostavno odkrivamo vozlišča. Postopek v prvem koraku je učinkovit, ima pa pomanjkljivost. Preciznost odkrivanja skrajšanih vektorjev je odvisna od kota pod katerim je narisana črta. Za kote od *45* do *135* stopinj (navpične črte) je natančnost zelo velika. Za kote od *-45* do *45* stopinj (horizontalne črte) natančnost ni zadovoljiva *(Slika 5.G)*.

Za rešitev tega problema smo našli enostavno in učinkovito rešitev. Rasterska slika, ki je zapisana s kodami tekočih dolžin, se prepiše v popolnoma enako novo sliko; samo je zapis kod za novo sliko drugačen. Pravila za zapis kod ostanejo ista, vendar se kode ne pišejo več po vrsticah, temveč po stolpcih, z leve proti desni. Slika vsebuje kode, ki so zarotirane za *90* stopinj.

Za novo, rotirano sliko isti postopek ponovimo. Vektorje vertikalnih črt originalne slike bomo dobili, če upoštevamo samo črte, ki so pod kotom od *45* do *135* stopinj (prvi del prveg koraka - obdelava nerotirane slike) *(Slika 6)*. Vektorje horizontalnih črt dobimo če upoštevamo samo črte, ki

so pod kotom od *45* do *135* stopinj, pri obdelavi rotirane slike (to so vektorji črt, ki so na originalni sliki pod kotom od - 5 do *45* stopinj) *(Slika 6)*. Celotna vektorizirana slika V je unija teh dveh podslik $V_{(45,135)}$ in $V_{(-45,45)}$. Prvi korak lahko ponazorimo z dvema preslikavama iz rasterskega R v vektorski prostor V:

$$R \longrightarrow V_{(45,135)}$$
$$R \longrightarrow V_{(-45,45)}$$
$$V = V_{(45,135)} \; U \; V_{(-45,45)}$$

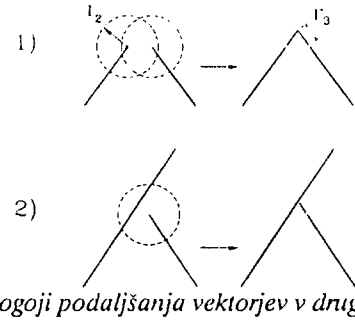


Slika 6. a)kopija dela črtne risbe
b)binarizirana slika
c)rezultat po prvem prehodu

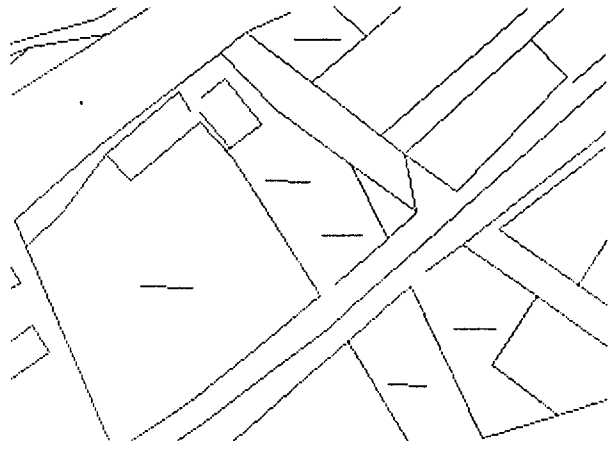
2. KORAK: Izgraditev vozlišč, prvi del

Za drugi korak je značilno da obdelava poteka samo na vektorski sliki, dobljeni v prvem koraku. Obdelava vektorske slike je bistveno hitrejša od obdelave rasterske slike. Po prvem koraku ostane še zmeraj problem za črte, ki so približno pod kotom *45* oz. *135* stopinj. Za te kote obstaja možnost podvojitve vektorjev zato, ker je lahko npr. črta, ki je pod kotom *45* stopinjah vektorizirana na nerotirani in tudi na rotirani sliki. Zato v drugem koraku vse vektorje, ki so približno pod kotoma *45* in *135* stopinj na nerotirani sliki, testiramo če so enaki (s predpisanim odstopanjem) na rotirani sliki. V slučaju obstoja takih parov odstranimo po eno črto iz para. S tem je predobdelava vektorjev v drugem koraku končana.

V glavnem delu drugega koraka testiramo vsak vektor, če obstaja v njegovi vnaprej določeni bližini, r_2 konec drugih vektorjev. To je večinoma slučaj na mestih preskočenih vozlišč, združevanja dveh ali treh črt in redkeje štirih ali več. V primeru pozitivnega testa, podaljšamo odkrite vektorje do presečišča. Če je podaljšek krajši od naslednje pragovne vrednosti r_3, vpišemo nov vektor na mesto starega. Na ta način zgradimo večino vozlišč. Na *Sliki 7.* so prikazane različne možnosti podaljševanja vektorjev v vozlišča.



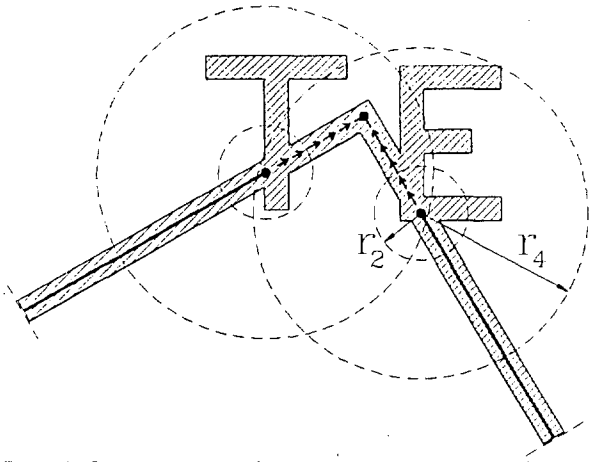
Slika 7. Pogoji podaljšanja vektorjev v drugem koraku



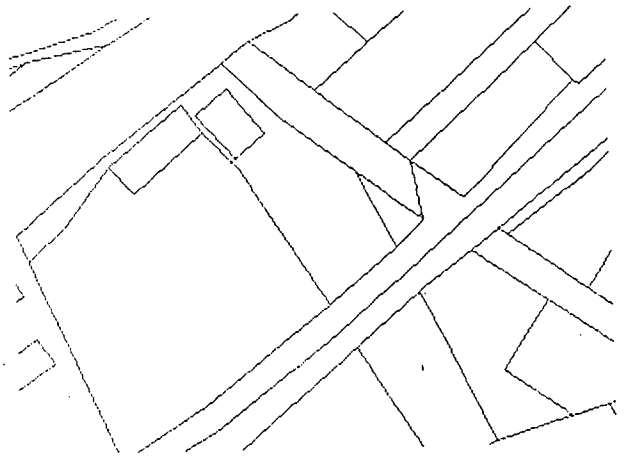
Slika 8. Rezultat po drugem prehodu

3. KORAK: Gradnja vozlišč, drugi del

Po drugem koraku lahko ostane manjši del vozlišč nezaključen. Ta primer je največkrat zaradi relativno majhne pragovne vrednosti r_2. V drugem koraku r_2 ne smemo povečati, ker bi lahko prišlo do nepravilne interpretacije rasterske slike.



Slika 9. Pogoji za podaljšanje vektorjev v tretjem koraku

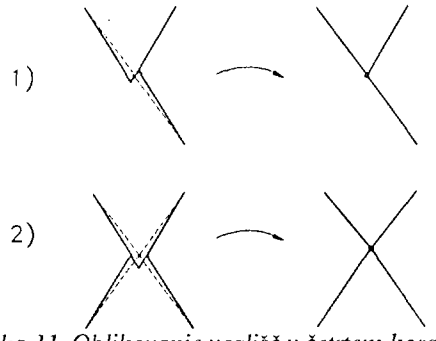


Slika 10. Rezultat po tretjem koraku

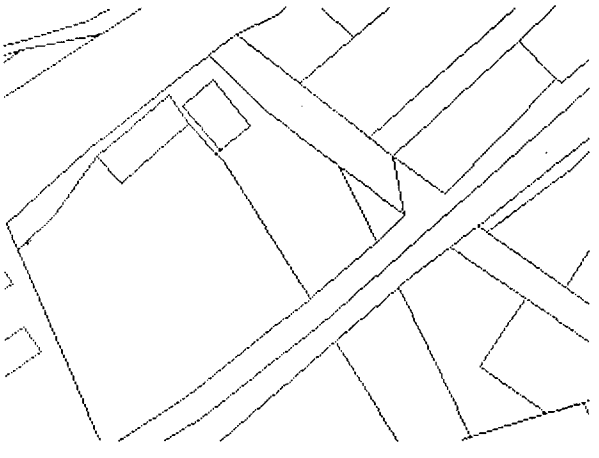
Zato v tretjem prehodu kombinirano testiramo vektorsko in rastersko sliko, vendar samo lokalno. Določimo novo pragovno vrednost r_4, ki je ponavadi dvakrat večja od r_2. Testiramo preostale nepodaljšane vektorje pod istimi pogoji kot v drugem prehodu (samo z r_4 namesto r_2). Če najdemo v področju r_4 še eno ali več nepodaljšanih vektorjev, potem začasno podaljšamo vse te vektorje do presečišča. Na rasterski sliki testiramo, če so novi vektorji podaljšani preko rasterskih črt *(Slika 9)*. Če je test pozitiven, lahko vpišemo nove podaljšane vektorje na mesto starih.

4. KORAK: Oblikovanje vozlišč

Vozlišča, ki združujejo tri ali več črt in so bila zgrajena v drugem in tretjem prehodu, največkrat nimajo popolnoma pravilne oblike. Možne oblike nepravilnosti so povečano prikazane na *Sliki 11.* Te nepravilnosti so zelo majhne, registrirajo pa jih informacijski sistemi, ki uporabljajo vektorski zapis kot vhodne podatke. Zato v četrtem prehodu preverimo vsa vozlišča in jih ustrezno oblikujemo, kot je to prikazano na *Sliki 11.* Po četrtem koraku je postopek vektorizacije končan.



Slika 11. Oblikovanje vozlišč v četrtem koraku



Slika 12. Rezultat po četrtem koraku

Zaključek

Geografski Informacijski Sistemi *(GIS)* so računalniška orodja, ki zagotavljajo učinkovito vodenje, povezovanje in vrednotenje geometrijskih podatkov o okolju. Osnovni problem za izgradnjo *GIS*-ov je vnos podatkov v računalnik.

Večina podatkov o okolju je še vedno na papirju. Obstajajo različni ročni postopki vnosa teh podatkov v računalnik. V nalogi so preučeni postopki za digitalizacijo in vektorizacijo črtnih risb velikih dimenzij, katerih struktura opisuje metriko prostora. Ti postopki so avtomatizirani. Vnos podatkov je tako lažji in hitrejši.

Postopek vektorizacije črtnih risb je v veliki meri odvisen od vrste in vsebine črtne risbe, ki jo obdelujemo. V različnih okoljih so postavljeni različni standardi. V članku je predstavljena lastna metoda digitalizacije in vektorizacije, ki je zasnovana na zahtevah uporabnikov slovenskega prostora.

Postopek digitalizacije je priprava rasterske slike za nadaljno obdelavo s postopkom vektorizacije. Vektorizacija se izvaja v štirih zaporednih korakih. Izpolnjuje tri osnovne pogoje za uspešnost: natančnost rezultatov,

sprejemljivost dobljenih rezultatov in hitrost postopka. Na posebej izdelani testni sliki je bil preverjen naš postopek za prenos grafične informacije iz predloge v vektorski zapis. Testiranje je pokazalo, da naša metoda izpolnjuje vse tri pogoje za uspešno vektorizacijo.

Literatura

[1] A. Dedić, R. Murn, D. Peček, Postopki za obdelavo slik, *YUROB'89,* 1989, pp. 4.63-4.67.

[2] A. Dedić, R. Murn, D. Peček, Optični zajem in digitalizacija grafičnih predlog večjih dimenzij, *YUROB'90,* 1990, pp. 3.38-3.42

[3] A. Dedić, R. Murn, D. Peček, Digitalization of Large Area Drawings and Maps, *Melecon'91, IEEE Mediterranean Conf,* 1991, pp. 1260-1263.

[4] M.T.Musavi at al., A vision Based Method to Automate Map Processing, *Pattern Rocognition,* 1988, pp. 319-326.

[5] M. Ejiri at al., Automatic recognition of design and maps, *Proc. 7th Int Conf. Pattern Recognition,* pp. 1296-1305 (1984).

[6] N. Otsu at al, A threshold selection method from gray-level histograms. *IEEE Trans. Systems Man Cybernet.* 9, 1979, 62-66.

[7] Y. Nakagawa, Some experiments on variable thresholding, *Pattern recognition* 11, 1979, 191-204

[8] R. Esplid at al., A Raster-to-Vector-Conversation System Producing High Quality Geometric Entities, *The 6th Scandin. Conf. on Pattern Rec.,* 1989, 19-22

[9] S. Suzuki, Graph-Based Vectorization Method for Line Patterns, *Int. Conf. on Pattern Recognition,* 1988, pp. 616-621.

[10] S. Suzuki at al., Automatic line drawing Recognition of Large-Scale Maps, *Optical Eng.* 7, 1987, pp. 642-649

[11] K. Ramachandran, Coding Method for Vector Representation of Engineering Drawings, *Proceedings of the IEEE* 68, 1980, pp. 813-816.

UPORABA PREDZNANJA V INDUKTIVNEM UČENJU

INFORMATICA 4/91

Keywords: machine learning, inductive learning, background knowledge, inductive logic programmnig

Nada Lavrač
Istitut Jožef Stefan

Večina uveljavljenih metod za avtomatsko zajemanje znanja temelji na avtomatskem učenju iz primerov in uporablja atributni jezik za predstavitev primerov in naučenih konceptov. Atributni jezik je zelo omejen, saj ne omogoča opisovanja kompleksnih strukturiranih objektov ter relacij med objekti. V zadnjem času gre vse več pozornosti sistemom za avtomatsko učenje, ki uporabljajo izraznejše jezike prvega reda za predstavitev naučenih konceptov. Ti jeziki omogočajo uporabo relacij, ki izražajo predznanje o učnih primerih in domeni sami. Induktivno avtomatsko učenje relacij imenujemo tudi induktivno logično programiranje. V članku opišemo različne načine uporabe predznanja v atributnem učenju in v učenju relacij. Opišemo tudi uporabo predznanja v sistemu induktivnega logičnega programiranja LINUS. Na kratko predstavimo dve aplikaciji tega sistema.

THE USE OF BACKGROUND KNOWLEDGE IN INDUCTIVE CONCEPT LEARNING

Inductive concept learning is most frequently used for automatic knowledge acquisition. Most of the successful inductive learning methods use a propositional attribute-value language for the representation of training examples and concepts. This language is limited and does not allow for representing complex structured objects and relations among objects. The goal of a new research area named Inductive Logic Programming (ILP) is to develop systems which can learn relational descriptions of concepts in a first-order language of logic programs. A first-order language allows for describing relations which express experts' background knowledge about training examples and the domain as a whole. The paper describes the ways of using background knowledge in inductive concept learning, both in learning attribute and relational descriptions. Furthermore, the ways of using background knowledge in the system LINUS is described. LINUS is an integrated inductive learning environment which can be used for learning attribute descriptions using background knowledge, as well as for inductive logic programming. Two applications of LINUS are briefly described: learning rules for early diagnosis of rheumatic diseases and learning illegal chess endgame positions.

1. Uvod

V svojem razvoju je umetna inteligenca prišla do stopnje, ko so njene metode, tehnike in orodja postale splošno uporabne v raznovrstnih računalniških aplikacijah. Med njimi so najbolj znani, najuspešnejši in zato tudi komercialno najbolj zanimivi *ekspertni sistemi* (Jackson 1990), ki v izvrševanju nalog na ozkem problemskem področju dosegajo raven vrhunskih strokovnjakov - ekspertov. 'Inteligenca' ekspertnega sistema temelji na bazi znanja, specifični za konkretno problemsko področje. Proces izgradnje baze znanja je najtežavnejša faza v razvoju ekspertnega sistema in je znana pod imenom 'Feigenbaumovo ozko grlo'. Dolgotrajnost in težavnost konstrukcije baz znanja sta bila povod za številne

raziskave, katerih cilj je olajšati in pospešiti proces zajemanja znanja za ekspertne sisteme.

Večina uveljavljenih metod za avtomatsko zajemanje znanja temelji na *avtomatskem učenju iz primerov.* V tem pristopu iz danih primerov ekspertnih odločitev z induktivnim sklepanjem izpeljemo splošna pravila. Tudi sam ekspert včasih najlaže posreduje svoje znanje s pomočjo dobro izbranih primerov.

Večina praktično uporabnih sistemov za avtomatsko učenje, kot na primer ASISTENT (Cestnik, Kononenko in Bratko 1987), uporablja *atributni jezik* za predstavitev primerov in naučenih konceptov (opisov razredov). Pri teh sistemih so primeri predstavljeni z n-tericami vrednosti atributov ter ustreznim razredom, medtem ko so naučeni koncepti opisani z odločitvenimi drevesi ali z if-then pravili. Vse informacije o domeni so vsebovane v učnih primerih. Atributni jezik je zelo omejen, saj ne omogoča opisovanja kompleksnih strukturiranih objektov ter relacij med objekti. Zato obstaja vrsta nalog, ki jih z atributnim učenjem iz primerov ni mogoče rešiti.

S problemom omejenosti opisnega jezika se lahko spopademo na več načinov:

- Metode za *konstruktivno učenje* omogočajo, da sistem avtomatsko konstruira nove, sestavljene izraze. V interakciji s sistemom ekspert izbere in poimenuje tiste od predlaganih avtomatsko konstruiranih izrazov, ki bi jih bilo smiselno uporabiti v učenju opisov konceptov.

- Nove izraze lahko na osnovi svojega znanja o problemski domeni predlaga ekspert. Ekspert lahko torej poda svoje *predznanje* (angl. background knowledge) kot funkcije vrednosti obstoječih atributov lahko pa poda tudi smiselne relacije med objekti problemskega prostora.

- Moč atributnega jezika lahko povečamo tako, da izberemo izraznejši logični jezik prvega reda za predstavitev naučenih konceptov.

V zadnjem času gre vse več pozornosti sistemom za avtomatsko učenje, ki uporabljajo izraznejše jezike prvega reda za predstavitev naučenih konceptov. Ti jeziki omogočajo uporabo *relacij*, ki izražajo *predznanje* o učnih primerih in domeni sami. Medtem ko je v atributnih jezikih naloga učenje opisa razredov iz učnih primerov, je v relacijskih jezikih naloga učenje *logičnih definicij relacij* iz primerov ter predznanja. V relacijskem učenju so učni primeri opredeljena dejstva oz. n-terice vrednosti argumentov relacije, katere definicije se hočemo naučiti. Po analogiji z razredom je vsak učni primer označen z $\oplus$, če pripada, oz. z $\ominus$, če ne pripada dani relaciji. Predznanje je tudi izraženo v obliki relacij, ki so podane s pripadajočimi n-tericami, ali pa z logičnimi definicijami (programi) v programskem jeziku *prolog.* Ker so naučene definicije relacij tudi izražene kot *logični programi*, induktivno avtomatsko učenje relacij imenujemo tudi *induktivno logično programiranje* (Muggleton 1991).

V članku opišemo različne načine uporabe predznanja v atributnem učenju in v učenju relacij. V drugem razdelku definiramo problem induktivnega učenja konceptov. S primeri predstavimo naloge atributnega učenja, atributnega učenja z upoštevanjem predznanja ter učenja relacij v induktivnem logičnem programiranju. V tretjem razdelku obravnavamo uporabo predznanja v sistemu induktivnega logičnega programiranja LINUS. Na kratko opišemo algoritem in dve aplikaciji sistema LINUS: učenje medicinskih diagnostičnih pravil v revmatologiji ter učenje relacij v šahovski končnici.

2. Induktivno učenje konceptov

Koncept C lahko definiramo kot podmnožico prostora objektov. Induktivno učenje konceptov pomeni metodo učenja iz primerov, ki omogoča razpoznavanje objektov v C.

Učni primer za učenje koncepta C je par:

$$\langle Objekt, Razred \rangle$$

kjer je *Objekt* izbrani opis objekta, *Razred* pa $\oplus$ ali $\ominus$, kar označuje, da *Objekt* pripada ali ne pripada konceptu C. Učni primer je *pozitiven*, če *Objekt* pripada konceptu C ($Razred = \oplus$) in je *negativen* primer ali *protiprimer*, če ne pripada C ($Razred = \ominus$). Namesto da bi dani objekt klasificirali v enega od dveh razredov $\oplus$ and $\ominus$, ga lahko klasificiramo v enega od več medsebojno izključujočih se razredov C_c, $c = 1, \ldots, N$.

Problem induktivnega učenja konceptov lahko formalno definiramo takole (de Raedt 1991):

	A_1	A_2	A_3	...	*Razred*
$primer_1$	v_{11}	v_{12}	v_{13}	...	C_1
$primer_2$	v_{21}	v_{22}	v_{23}	...	C_2
$primer_3$	...	...	...	...	...

Tabela 1: Množica primerov, zapisana v obliki tabele.

Podani so:

- jezik za opis primerov L_E
- množica pozitivnih primerov P
- množica negativnih primerov N
- jezik za opis konceptov L_C
- relacija pokritosti med L_C in L_E
- kriterij kvalitete definiran na nizih iz L_C

Poišči:

- opis koncepta C iz L_C, ki zadošča kriteriju kvalitete

Kriterij kvalitete lahko zahteva, da je zgrajeni opis koncepta konsistenten in popoln glede na dano množico primerov. Opis koncepta je *konsistenten*, če ne pokriva nobenega negativnega primera. Opis je *popoln*, če pokriva vse pozitivne primere. Zahtevo po konsistenstnosti in popolnosti moramo omiliti, ko se učimo iz šumnih in nepopolnih podatkov. Tako lahko preprečimo, da bi sistem zgradil preveč specifične opise konceptov.

Glede na izbor jezika za opis konceptov in primerov ločujemo dve glavni področji v induktivnem učenju: *atributno učenje* in *učenje relacij.*

2.1 Induktivno učenje atributnih opisov konceptov

V *atributnem učenju* so primeri predstavljeni z n-tericami vrednosti atributov ter z ustreznim razredom. Objekt lahko pripada enemu od N medsebojno izključujočih se razredov C_c. Množico primerov $\langle Objekt, Razred \rangle$ lahko predstavimo s tabelo 1.

Naloga atributnega učenja je poiskati klasifikacijsko *pravilo (hipotezo, opis koncepta)* kot funkcijo vrednosti atributov, ki ga lahko uporabimo za napovedovanje razreda neznanega objekta. Naučeni opis koncepta ima lahko obliko if-then pravila:

A1	*A2*	*A3*	*A4*	*A5*	*Razred*
da	*balon*	*da*	*kvadrat*	*kvadrat*	*P*
da	*zastava*	*da*	*osmerokotnik*	*osmerokotnik*	*P*
da	*meč*	*da*	*krog*	*osmerokotnik*	*N*
da	*meč*	*ne*	*kvadrat*	*osmerokotnik*	*N*
ne	*meč*	*ne*	*osmerokotnik*	*krog*	*N*
ne	*zastava*	*ne*	*krog*	*osmerokotnik*	*N*

Tabela 2: Primeri prijaznih (P) in neprijaznih (N) robotov.

$$\begin{array}{ll} if & (A_i = v_i) \wedge (A_j = v_j) \wedge \ldots \\ then & Razred = C_c \end{array}$$

ki je konsistentno in popolno glede na dano množico primerov.

Naučena if-then pravila opisujejo posamezne razrede. Sistemi pa se lahko naučijo tudi opisa vseh razredov v obliki *odločitvenega drevesa*, katerega listi označujejo posamezne razrede.

Za ilustracijo učenja odločitvenih dreves izberimo primer iz sveta prijaznih in neprijaznih robotov (Wnek, Sarma, Wahab in Michalski 1990, Sutlič 1991). Opis robota je podan z naslednjimi atributi in njihovimi vrednostmi:

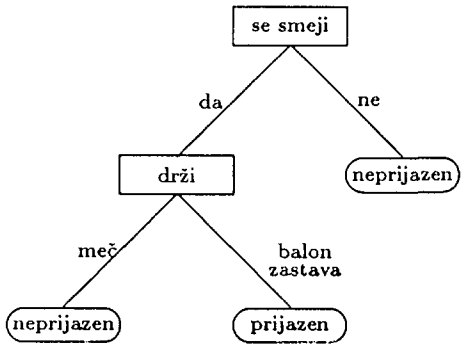
A1 - se smeji:	*da, ne*
A2 - drži:	*balon, zastava, meč*
A3 - ima kravato:	*da, ne*
A4 - oblika glave:	*krog, kvadrat, osmerokotnik*
A5 - oblika telesa:	*krog, kvadrat, osmerokotnik*

V tabeli 2 je podanih šest učnih primerov. Za vsako 5-terico vrednosti atributov je podan razred, ki določa, ali je robot prijazen ali ne.

Naloga učenja je poiskati pravilo, ki opredeli, ali je robot prijazen ali ne. Za učenje odločitvenih dreves lahko uporabimo sistem ASISTENT (Cestnik, Kononenko in Bratko 1987). Naučeno odločitveno drevo je prikazano na sliki 1.

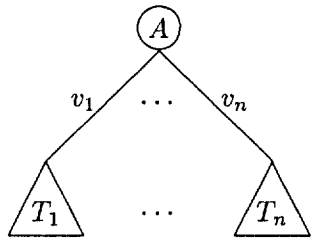
Opišimo Quinlanov algoritem ID3 (Quinlan 1986), ki je osnova algoritma ASISTENT. Naj bo S množica učnih primerov, $C_1, \ldots C_N$ pa odločitveni razredi.

- *če* vsi primeri iz S spadajo v isti razred C_c
- *potem* označi list drevesa s C_c
- *sicer*



Slika 1: Odločitveno drevo, zgrajeno s sistemom ASISTENT iz šestih primerov robotov.

- izberi 'najinformativnejši' atribut A z vrednostmi $v_1, \ldots, v_n$
- razdeli učno množico S v $S_1, \ldots, S_n$ glede na vrednosti $v_1, \ldots, v_n$
- rekurzivno zgradi poddrevesa $T_1, \ldots, T_n$ za $S_1, \ldots, S_n$
- zgradi odločitveno drevo T, kot je prikazano na sliki 2.



Slika 2: Odločitveno drevo, ki ga zgradi ID3.

Preiskovalna hevristika algoritma je določena z *informativnostjo* atributa, ki se izračuna na osnovi *entropije* učne množice $E(S)$, ki izraža količino informacije, potrebne za klasifikacijo enega objekta:

$$E(S) = -\sum_{c=1}^{N} p_c \cdot log_2 p_c$$

kjer je p_c apriorna verjetnost razreda C_c (izračunana z relativno frekvenco razreda C_c v S).

Entropija učne množice po razbitju po vrednostih $v_1, \ldots, v_n$ atributa A je:

$$E(S/A) = -\sum_{v=1}^{n} p_v \cdot \sum_{c=1}^{N} \frac{p_{vc}}{p_v} \cdot log_2 \frac{p_{vc}}{p_v}$$

kjer je p_v apriorna verjetnost, da ima A vrednost v_v, p_{vc} pa je apriorna verjetnost, da ima primer, ki spada v razred C_c, vrednost v_v atributa A.

Informativnost atributa I(A) je mera, ki minimizira število testov, potrebnih za klasifikacijo novega objekta: $I(A) = E(S) - E(S/A)$. *Najinformativnejši atribut* je tisti, pri katerem doseže informativnost maksimum $max\ I(A)$, kar je ekvivalentno minimizaciji entropije $min\ E(S/A)$.

V sistemu ASISTENT je bil algoritem ID3 izboljšan na več načinov. Omogoča obravnavo manjkajočih podatkov, obravnavo numeričnih (zveznih) vrednosti atributov, obravnavo NULL listov, v katere ne spada noben učni primer, obravnavo SEARCH listov, v katere spadajo primeri iz več različnih razredov, binarizacijo atributov, ter obravnavo netočnih podatkov, tako da vpelje rezanje dreves; rezanje omogoča izgradnjo manjših drevesa, ki so lažje razumljiva in imajo večjo klasifikacijsko točnost.

Drugi primeri sistemov, ki se učijo opisov konceptov v obliki odločitvenih dreves so npr. C4 (Quinlan et al. 1986) in CART (Breiman et al. 1984). Primeri sistemov, ki se učijo if-then pravil pa so AQ15 (Michalski et al. 1986), njegov predhodnik NEWGEM (Mozetič 1985), CN2 (Clark in Niblett 1989) ter GINESYS (Gams 1989).

2.2 Uporaba predznanja v atributnem učenju

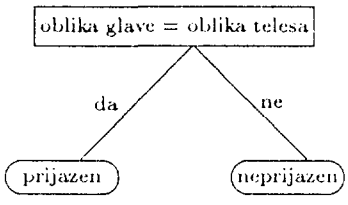
Kot smo omenili v uvodu, lahko sistem v naučenih pravilih upošteva nove izraze, ki mu jih na osnovi svojega znanja o problemski domeni predlaga ekspert.

Pri odločanju in pri razlagi odločitev eksperti namreč uporabljajo svoje znanje o problemu. To znanje je mogoče izraziti kot funkcije vrednosti atributov (s katerimi so opisani učni primeri) ali pa z relacijami med atributi ali objekti problemskega področja. To ekspertovo znanje, imenovano *predznanje* (angl. background knowledge), lahko sistem koristno uporabi pri učenju kompleksnejših relacij. Predznanje lahko sistem uporabi kot pomožno znanje v atributnem učenju. V učenju relacij pa postane relacijsko predznanje bistvenega pomena za učenje.

V atributnem učenju sistem upošteva predznanje tako, da za vsak nov (sestavljen) izraz, podan v predznanju, vpelje nov atribut. Če poda ekspert svoje znanje v funkcijski obliki, se za posamezni

A1	*A2*	*A3*	*A4*	*A5*	*A6*	*Razred*
da	*balon*	*da*	*kvadrat*	*kvadrat*	*da*	*P*
da	*zastava*	*da*	*osmerokotnik*	*osmerokotnik*	*da*	*P*
da	*meč*	*da*	*krog*	*osmerokotnik*	*ne*	*N*
da	*meč*	*ne*	*kvadrat*	*osmerokotnik*	*ne*	*N*
ne	*meč*	*ne*	*osmerokotnik*	*krog*	*ne*	*N*
ne	*zastava*	*ne*	*krog*	*osmerokotnik*	*ne*	*N*

Tabela 3: Primeri prijaznih (P) in neprijaznih (N) robotov. V stolpcu $A6$ so podane vrednosti novega atributa *oblika glave = oblika telesa.*



Slika 3: Odločitveno drevo, zgrajeno s sistemom ASISTENT iz šestih primerov robotov z uporabo predznanja.

učni primer vrednost novega atributa izračuna kot funkcija vrednosti osnovnih atributov. Če pa je predznanje podano v obliki relacij, ima novo vpeljani atribut vrednost *true*, v kolikor vrednosti ustreznih atributov primera ustrezajo relaciji, sicer ima vrednost *false*.

V primeru iz sveta robotov vpeljimo en sestavljen izraz, s katerim testiramo relacijo enakosti: *oblika glave = oblika telesa.* S tem smo vpeljali nov atribut $A6$ z naborom vrednosti *da* (*true/resnično*) in *ne* (*false/ni resnično*). Množico učnih primerov, dopolnjeno z vrednostmi novega atributa $A6$, prikažemo v tabeli 3.

V gradnji odločitvenega drevesa iz primerov podanih v tabeli 3 se novi atribut $A6$ izkaže kot najbolj informativen, zato se pojavi v korenu drevesa zgrajenega s sistemom ASISTENT. Ker vrednosti atributa $A6$ v celoti ločita med prijaznimi in neprijaznimi roboti, je to tudi edini atribut v zgrajenem drevesu. Drevo je podano na sliki 3.

Iz primera je razvidno, da je lahko atribut, ki je zgrajen na osnovi predznanja, bolj informativen kot so osnovni atributi. Z uporabo predznanja lahko zgradimo kompaktnejše opise konceptov, ki imajo lahko večjo klasifikacijsko točnost.

2.3 Induktivno logično programiranje

V *učenju relacij* učni primeri izražajo relacije med elementi problemskega prostora (objekti), podano pa je tudi dodatno problemsko znanje (predznanje) o učnih primerih in domeni sami. Medtem ko je v atributnih jezikih naloga učenje opisov konceptov (razredov) iz učnih primerov, je v relacijskih jezikih naloga učenje logičnih definicij relacij iz primerov in predznanja.

Nalogo učenja relacij v jeziku prvega reda lahko formuliramo takole:

Podani so:

- jezik za opis primerov L_E
- množica pozitivnih in negativnih primerov (dejstev $\oplus$ in $\ominus$) neznane relacije p
- jezik za opis konceptov L_C, ki določa sintaktične omejitve za definicijo predikata p
- jezik za opis predznanja L_B
- predznanje - množica definicij pomožnih predikatov in funkcij q_i, ki jih lahko uporabimo v definiciji predikata p
- relacija pokritosti med L_C in L_E glede na L_B
- kriterij kvalitete definiran na definicijah predikatov iz L_C

Poišči:

- definicijo ciljnega predikata p kot množico stavkov iz L_C, ki zadošča kriteriju kvalitete

Množica pozitivnih ($\oplus$) in negativnih ($\ominus$) primerov je podana z opredeljenimi dejstvi (angl. ground facts), kar pomeni, da so podane vrednosti argumentov relacije, t.j., da v argumentih relacije ne nastopajo spremenljivke. Predznanje je podano z funkcijami in relacijami, ki vsebujejo informacijo o argumentih ciljne relacije p. Naloga učenja je poiskati definicijo relacije p v odvisnosti od relacij iz predznanja, ki bo ustrezala kriteriju kvalitete, npr. ki bo popolna in konsistentna glede na dana dejstva. Definicija je popolna, če razlaga (pokriva) vsa pozitivna dejstva, konsistentna pa, če ne pokriva nobenih negativnih dejstev.

Naučena definicija predikata p sestoji iz množice stavkov oblike

$$p(X_1, X_2, \ldots, X_k) \leftarrow L_1, \ldots, L_n$$

kjer je telo stavka konjunkcija literalov $L_i = q_i(Y_1, Y_2, \ldots, Y_m)$.

Ilustrirajmo zgornjo definicijo na preprostem problemu učenja družinskih relacij (Džeroski 1991). Naj bo ciljna relacija *daughter* in naj bo predznanje sestavljeno iz relacij *female* in *parent*. Za relacijo *daughter* so podani štirje učni primeri: dva pozitivna in dva negativna.

$$\begin{array}{ll} daughter(maja, ana). & \oplus \\ daughter(eva, tom). & \oplus \\ daughter(tom, ana). & \ominus \\ daughter(eva, ana). & \ominus \end{array}$$

Podano je naslednje predznanje:

$$\begin{array}{ll} parent(ana, maja). & female(ana). \\ parent(ana, tom). & female(maja). \\ parent(tom, eva). & female(eva). \\ parent(tom, ivo). & \end{array}$$

Iz učnih primerov in predznanja se je mogoče naučiti definicije predikata *daughter*:

$$daughter(X, Y) \leftarrow female(X), parent(Y, X).$$

Naučene definicije relacij so podane s programskimi stavki v sintaksi programskega jezika prolog. Definicijo lahko v kompleksnejših primerih sestavlja več programskih stavkov (angl. program clauses). Množico stavkov z istim predikatom v glavi stavka imenujemo *definicija predikata* (angl. predicate definition). Definicija predikata tvori prologovo proceduro oziroma prologov logični program.

Sistemi, ki se učijo logičnih programov, spadajo med sisteme *induktivnega logičnega programiranja (ILP)*. Induktivno logično programiranje je mlado raziskovalno področje. V svetu so bile prve raziskave s tega področja opravljene leta 1981, ko je bil razvit sistem MIS (Model Inference System, Shapiro 1981), ki se zna učiti logičnih programov, temu pa so sledili sistemi MARVIN (Sammut in Banerji 1986), CIGOL (Muggleton in Buntine 1988), GOLEM (Muggleton in Feng 1990) in FOIL (Quinlan 1990). Področje je dobilo svoje ime šele leta 1990, prva delavnica (First International Workshop on Inductive Logic Programming) pa je bila marca 1991.

Področje ILP je razvito tudi v Sloveniji. Razvita sta bila ILP sistema LINUS (Lavrač 1990) in mFOIL (Džeroski 1991), ki sta bila uporabljena v več domenah, znanih iz literature o avtomatskem učenju (Lavrač, Džeroski in Grobelnik 1991), za učenje medicinskih diagnostičnih pravil na področju revmatologije (Lavrač, Džeroski, Pirnat in Križman 1991), za učenje načrtovanja mrež končnih elementov (Džeroski in Dolšak 1991) ter za učenje iz šumnih podatkov (Lavrač in Džeroski 1991, Džeroski in Lavrač 1991a). Teoretično primerjavo sistemov LINUS in FOIL smo podali z grafi specializacije (Džeroski in Lavrač 1991b). ILP pristop je bil uporabljen tudi za učenje kvalitativnih modelov (Bratko, Muggleton in Varšek 1991).

3. Uporaba predznanja v integriranem okolju za avtomatsko učenje LINUS

Sistem induktivnega logičnega programiranja LINUS (Lavrač 1990, Lavrač, Džeroski in Grobelnik 1991) temelji na ideji, da je možno transformirati problem učenja relacij v problem atributnega učenja z uporabo predznanja. Na ta način lahko uporabimo obstoječe atributne učne algoritme, npr. sistem ASISTENT, ki so že dosegli veliko stopnjo uporabnosti in znajo obravnavati nepopolne in šumne podatke.

Sistem LINUS omogoča uporabo dodatnega problemskega znanja v učenju opisov konceptov (opisov posameznih razredov objektov) ali učenju logičnih definicij relacij med objekti problemskega prostora. V sistemu LINUS opišemo učne primere in naučena pravila v jeziku *deduktivnih hierarhičnih baz podatkov* (DHBP), t.j. v logičnem jeziku prega reda, ki je izraznejši od atributnih jezikov 0-tega reda, ki jih uporablja večina uveljavljenih sistemov za induktivno učenje. Izbor tega preprostega jezika prvega reda omogoča sistemu LINUS, da z uporabo atributnih učnih algoritmov učinkovito zgradi logične opise relacij.

Stavki DHBP so omejena oblika programskih stavkov; prepovedani so rekurzivni stavki, spremenljivkam pa so pridružene ustrezne končne zaloge vrednosti (tipi). Dodatna omejitev LINUSu onemogoča uvajanje novih spremenljivk v telo stavka. Predznanje je pri LINUSu izraženo v formalizmu *deduktivnih baz podatkov* (DBP), ki dovoljuje rekurzivne definicije predikatov.

Sistem LINUS vključuje tri atributne učne algoritme ASISTENT, NEWGEM in CN2. ASISTENT je predstavnik TDIDT (Top-Down Induc-

tion of Decision Trees) družine učnih algoritmov (Quinlan 1986). NEWGEM je član družine AQ algoritmov (npr. Michalski et al. 1986). CN2 združuje najboljše lastnosti ID3 in AQ algoritmov s tem, da dovoli uporabo statističnih metod, podobnih rezanju drevesa, in da generira if-then pravila.

3.1 Opis algoritma LINUS

Program, ki omogoča vključitev sistema LINUS v okolje logičnega programiranja imenujemo DHBP vmesnik. Program je implementiran v prologu. Vmesnik pretvori pozitivna dejstva in negativna dejstva (negativna dejstva so podana ali pa jih generira DHBP vmesnik) iz DHBP oblike v obliko n-teric vrednosti atributov, atributni učni algoritem se nauči opisa koncepta v atributnem jeziku, DHBP vmesnik pa pretvori te opise v obliko DHBP stavkov. Najpomembnejši del DHBP vmesnika je generiranje novih atributov za učenje. Z upoštevanjem zaloge vrednosti tipov argumentov ciljne relacije namreč preveri, katere so možne aplikacije pomožnih predikatov in funkcij iz predznanja ter le-te upošteva kot dodatne atribute, ki jih bo uporabil v učenju. Vsak učni primer dopolni z vrednostmi teh dodatnih atributov, to je z vrednostmi *da* (*true*) ali *ne* (*false*) za pomožne predikate, oziroma z vrednostmi izhodnih argumentov za pomožne funkcije. DHBP vmesnik torej implementira osnovno idejo uporabe predznanja v atributnem učenju, ki smo jo razložili v razdelku 2.2.

Učenje z LINUSom poteka v naslednjih korakih:

- pripravi množico pozitivnih in negativnih dejstev,
- pretvori dejstva iz DHBP oblike v n-terice vrednosti atributov,
- nauči se opisa koncepta z uporabo atributnega učnega algoritma,
- pretvori naučena pravila iz oblike if-then pravil v obliko DHBP stavkov.

Algoritem opišemo s preprostim primerom. Denimo, da je relacija $r(a,b,c)$ podana z naslednjimi tremi dejstvi, označenimi z $\oplus$:

$$r(a1,b1,b1).$$
$$r(a1,b2,b2).$$
$$r(a2,b1,b2).$$

Podana je tudi informacija o *imenih* in *tipih* spremenljivk. Prvi argument relacije r ima ime a, njegov tip je *tip_a*, drugi in tretji argument b in c pa sta oba istega tipa *tip_b*. Vsakemu tipu je pridružena informacija o zalogi vrednosti spremenljivk: $\{a1,a2\}$ za *tip_a* in $\{b1,b2\}$ za *tip_b*.

Dodatno znanje o problemu je podano v obliki definicij pomožnih predikatov in funkcij. Uporabimo npr. binarni pomožni predikat p, definiran z naslednjimi dejstvi:

$$p(a1,b1).$$
$$p(a1,b2).$$
$$p(a2,b2).$$

V splošnem so lahko pomožni predikati in funkcije definirani z množico rekurzivnih tipiziranih programskih stavkov (v formalizmu DBP). Argumenta predikata p sta tipa *tip_a* in *tip_b*. LINUS uporablja tudi vgrajeni pomožni predikat *enakost* (=/2).

Množica *negativnih dejstev* je lahko vnaprej podana, lahko pa jo sistem generira iz pozitivnih dejstev. V načinu, ki temelji na *predpostavki zaprtega sveta (cwa)*, LINUS generira vse možne kombinacije vrednosti argumentov ciljne relacije. Vsa generirana dejstva, razen podanih pozitivnih dejstev, LINUS uporabi kot negativna dejstva in jih označi z $\ominus$:

$$r(a1,b1,b2).$$
$$r(a1,b2,b1).$$
$$r(a2,b1,b1).$$
$$r(a2,b2,b1).$$
$$r(a2,b2,b2).$$

V drugem koraku algoritem preveri vse možne aplikacije pomožnih predikatov in funkcij glede na tipe spremenljivk. V našem primeru so to $p(a,b)$ in $p(a,c)$, enakost pa lahko uporabi nad argumentoma istega tipa, torej b=c. Iz treh pozitivnih primerov relacije r LINUS generira naslednje šesterice oblike:

$$\langle a, \quad b, \quad c, \quad (b=c), \quad p(a,b), \quad p(a,c) \rangle$$

$$\langle a1, \quad b1, \quad b1, \quad true, \quad true, \quad true \rangle$$
$$\langle a1, \quad b2, \quad b2, \quad true, \quad true, \quad true \rangle$$
$$\langle a2, \quad b1, \quad b2, \quad false, \quad false, \quad true \rangle$$

Na enak način LINUS generira šesterice tudi iz negativnih dejstev.

V tretjem koraku generirane n-terice pozitivnih in negativnih primerov uporabi za učenje s sistemi ASISTENT, NEWGEM ali CN2. Generirani opis koncepta v obliki odločitvenega drevesa, VL1 pravila ali if-then pravil, se prepiše v prologovo obliko if-then pravil.

V četrtem koraku algoritem pretvori if-then pravila iz prologove oblike v obliko DHBP stavkov. V našem preprostem primeru dobimo naslednji opis:

$$r(A,B,C) \leftarrow A = a1, B = C.$$
$$r(A,B,C) \leftarrow C = b2,\ not\ p(A,B).$$

V generiranih pravilih nastopata poleg literalov *A=a1* in *C=b2*, ki jih je mogoče dobiti tudi z atributnim učenjem, tudi literala *B=C* in *not p(A,B)*, ki sta dobljena z uporabo relacijskega predznanja.

Z izborom atributov in predznanja, ki naj se uporabi za učenje, lahko torej sistem LINUS uporabimo na tri načine:

- kot atributni učni algoritem, če sistemu povemo, kateri od argumentov ciljne relacije naj upošteva kot odločitveni razred ter če v celoti izključimo predznanje,
- kot algoritem za atributno učenje, ki zna uporabljati predznanje, če sistemu povemo, kateri od argumentov ciljne relacije naj upošteva kot odločitveni razred ter če upoštevamo nove atribute, dobljene z uporabo predznanja,
- kot algoritem za relacijsko učenje, če obravnavamo samo pozitivne in negativne primere relacije $\oplus$ in $\ominus$ ter se učimo samo na osnovi predznanja.

3.2 Uporaba predznanja v atributnem učenju opisov posameznih razredov v revmatologiji

LINUS predstavlja razširitev atributnih algoritmov z možnostjo uporabe predznanja in ga lahko uporabimo za atributno učenje. Takemu načinu učenja pravimo *učenje razredov* (class learning mode, Lavrač, Džeroski in Grobelnik 1991). Sistem smo v tem načinu uporabili za učenje diagnostičnih pravil v revmatologiji. Eksperimenti so podrobno opisani v (Lavrač, Džeroski, Pirnat in Križman 1991). Na voljo smo imeli podatke o 462 pacientih Univerzitetnega kliničnega centra v Ljubljani. Pacienti so bili razvrščeni v 200 diagnoz, ki so bile grupirane v 8 razredov diagnoz: *A1* degenerativne bolezni hrbtenice (158 primerov), *A2* degenerativne bolezni sklepov (128), *B1* vnetne bolezni hrbtenice (16), *B234* druge vnetne bolezni (29), *C* zunajsklepni revmatizem (21), *D* metabolni revmatizem (24), *E* neznačilne revmatske težave (32) ter *F* nerevmatske bolezni (54). V eksperimentu smo upoštevali le anamnestične podatke, opisane z vrednostmi naslednjih 16 atributov: spol, starost, družinska anamneza, sedanje težave, sedanja bolezen, bolečine v sklepih, število bolečih sklepov, število oteklih sklepov, bolečine v hrbtenici, ostale bolečine, jutranja okorelost, koža, sluznice, oči, druge težave in terapija.

LINUS smo uporabili za učenje opisov posameznih razredov diagnoz. 70 % podatkov smo uporabili za učenje, 30 % pa za testiranje klasifikacijske točnosti naučenih pravil. Eksperiment smo ponovili desetkrat, z naključno porazdelitvijo v učne in testne primere. V učenju smo upoštevali predznanje o značilnih skupinah simptomov. Identificirali smo 6 značilnih skupin simptomov, npr. karakteristične kombinacije simptomov prve skupine so naslednje:

Bolečine v sklepih	Jutranja okorelost
bp	$\leq$ 1 ura
artrotična	$\leq$ 1 ura
artritična	$>$ 1 ura

Karakteristične kombinacije v šesti skupini simptomov pa so podane v spodnji tabeli:

Število oteklih sklepov	Število bolečih sklepov
0	0
0	$1 \leq$ št. sklepov ≤ 30
$1 \leq$ št. sklepov ≤ 10	$0 \leq$ št. sklepov ≤ 30

Kot primer navedimo diagnostično pravilo za metabolni revmatizem:

```
Razred = metabolni_revmatizem ←
  skupina6( St_otekl_sklepov, St_bol_sklepov,
            '1=<os=<10&0=<bs=<30' ),
  Starost > 30, Starost =< 40,
  Spol = moski.

Razred = metabolni_revmatizem ←
  Bol_v_hrbtenici = bp,
```

Bol_v_sklepih = artriticne,
Spol = moski.

Razred = metabolni_revmatizem ←
Bol_v_hrbtenici = bp,
Starost =< 40,
Sedanje_tezave > 11,
Spol = moski,
skupina5(Bol_v_sklepih, Bol_v_hrbtenici,
St_bol_sklepov, Vrednost),
member(Vrednost, ['artriticne&bp&1=<bs=<30',
neznacilno]).

3.3 Uporaba predznanja v učenju relacij v šahovski končnici

Kot primer uporabe predznanja v učenju logičnih definicij relacij izberimo učenje nelegalnih pozicij v šahovski končnici beli kralj in bela trdnjava proti črnemu kralju. Različni eksperimenti v tej domeni so podrobno opisani v (Lavrač, Džeroski in Grobelnik 1991, Lavrač in Džeroski 1991, Džeroski in Lavrač 1991a).

Ciljna relacija je podana s predikatom *nelegalen(A,B,C,D,E,F)*. Argumenti označujejo pozicijo belega kralja (A, B), bele trdnjave (C, D) in črnega kralja (E, F).

Predznanje je podano z relacijami *sosedna_vrsta(X,Y)*, *soseden_stolpec(X,Y)*, *manjsa_vrsta(X,Y)*, *manjsi_stolpec(X,Y)* in *enakost* $X = Y$. Spremenljivke so tipizirane. Vpeljemo dva tipa: *vrsta* z vrednostmi $\{1,2,3,4,5,6,7,8\}$ in *stolpec* z vrednostmi $\{a,b,c,d,e,f,g,h\}$.

Za eksperimente smo imeli na voljo :

- 5 učnih množic po 100 primerov, testno množico 5000 primerov
- 5 učnih množic po 1000 primerov, 5 testnih množic po 4500 primerov

Iz množice 100 učnih primerov se je LINUS naučil naslednje definicije relacije *nelegalen*:

nelegalen(A,B,C,D,E,F) ← *C=E.*
nelegalen(A,B,C,D,E,F) ← *D=F.*
nelegalen(A,B,C,D,E,F) ← *B=F,*
soseden_stolpec(A,E),
nelegalen(A,B,C,D,E,F) ←
soseden_stolpec(A,E),
sosedna_vrsta(B,F).
nelegalen(A,B,C,D,E,F) ← *A=E,*
sosedna_vrsta(B,F).
nelegalen(A,B,C,D,E,F) ← *A=E, B=F.*

Rezultate učenja s sistemom LINUS smo primerjali z nekaterimi drugimi ILP sistemi. V petih ponovitvah eksperimenta na množicah s po 100 učnimi primeri je bila dosežena naslednja povprečna klasifikacijska točnost naučenih pravil: 98.1% LINUS z uporabo ASISTENTa, 88.4% LINUS z uporabo NEWGEMa. Na istih učnih in testnih množicah je CIGOL dosegel 77.2% točnost, FOIL pa 90.8% točnost naučenih pravil.

4. Zaključek

Članek poda pregled uporabe predznanja v induktivnem učenju konceptov. Predstavi metodo, s katero je mogoče sisteme za atributno učenje nadgraditi tako, da se z uporabo predznanja naučijo tudi relacijskih opisov konceptov. Z uporabo te metode smo zgradili integrirano okolje za avtomatsko učenje LINUS, ki uporablja obstoječe atributne učne algoritme za atributno učenje in za induktivno logično programiranje. V prispevku smo se osredotočili na možnosti uporabe predznanja za izgradnjo kompaktnejših atributnih opisov ter za učenje relacijskih opisov ter le na kratko opisali sam učni algoritem ter dve aplikaciji sistema LINUS.

Reference

Bratko, I., Muggleton, S. in Varšek, A. (1991) Learning qualitative models of dynamic systems. Eighth International Workshop on Machine Learning. Evanston, IL: Morgan Kaufmann.

Breiman, L., Friedman, J.H., Olshen, R.A. in Stone, C.J. (1984) Classification and regression trees. Belmont: Wadsworth.

Cestnik, B., Kononenko, I. in Bratko, I. (1987) ASSISTANT 86: A knowledge elicitation tool for sophisticated users. V: Bratko, I. in Lavrač, N. (ur.) Progress in machine learning. Wilmslow: Sigma Press.

Clark, P. in Niblett, T. (1989) The CN2 induction algorithm. Machine Learning 3 (4), 261-284. Kluwer Academic Publishers.

Džeroski, S. in Dolšak, B. (1991) Primerjava algoritmov za avtomatsko učenje relacij na problemu načrtovanja mrež končnih elementov. XXVI Jugoslovanska konferenca ETAN, Ohrid, Makedonija.

Džeroski, S. in Lavrač, N. (1991a) Learning relations from noisy examples: An empirical comparison of LINUS and FOIL. Eighth International Workshop on Machine Learning. Evanston, IL: Morgan Kaufmann.

Džeroski, S. in Lavrač, N. (1991b) Refinement graphs for FOIL and LINUS. V: Muggleton, S.H. (ur.) Inductive logic programming. Glasgow, UK: Academic Press (v tisku).

de Raedt, L. (1991) Interactive concept-learning. Doktorska disertacija, Department of Computer Science, Katholieke Universiteit Leuven, Belgium.

Džeroski, S. (1991) Handling noise in inductive logic programming. Magisterska naloga, Fakulteta za elektrotehniko in računalništvo, Univerza v Ljubljani.

Gams, M. (1989) New measurements highlight the importance of redundant knowledge. Fourth European Working Session on Learning, EWSL 89, 71-79. Montpellier, France: Pitman.

Jackson, P.J. (1990) Introduction to expert systems (Druga izdaja). Addison-Wesley.

Lavrač, N. (1990) Principles of knowledge acquisition in expert systems. Doktorska disertacija, Tehniška fakulteta, Univerza v Mariboru.

Lavrač, N., Džeroski, S. in Grobelnik, M. (1991) Learning nonrecursive definitions of relations with LINUS. Fifth European Working Session on Learning, EWSL 91. Porto, Portugal: Springer-Verlag.

Lavrač, N. in Džeroski, S. (1991) Inductive learning of relational descriptions from noisy examples. First Inductive Logic Programming Workshop, Viana de Castelo, Portugal.

Lavrač, N., Džeroski, S., Pirnat, V. in Križman, V. (1991) Learning rules for early diagnosis of rheumatic diseases. Third Scandinavian Conference on Artificial Intelligence, SCAI'91. Roskilde, Denmark: IOS Press.

Michalski, R.S., Mozetič, I., Hong, J. in Lavrač, N. (1986) The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. Fifth National Conference on Artificial Intelligence, AAAI-86, 1041-1045. Philadelphia, PA: Morgan Kaufmann.

Mozetič, I. (1985) NEWGEM: Program for learning from examples - Technical documentation and user's guide. Technical Report, University of Illinois at Urbana-Champaign, Department of Computer Science. Tudi: Delovno poročilo IJS-DP-4390, Institut Jožef Stefan, Ljubljana.

Muggleton, S.H. (1991) Inductive logic programming. New Generation Computing 8 (4), 295-318.

Muggleton, S.H. in Buntine, W. (1988) Machine invention of first-order predicates by inverting resolution. Fifth Int. Conference on Machine Learning, 339-352. Ann Arbor, MI: Morgan Kaufmann.

Muggleton, S.H. in Feng, C. (1990) Efficient induction of logic programs. First Conference on Algorithmic Learning Theory. Tokyo: Ohmsha.

Quinlan, J.R. (1986) Induction of decision trees. Machine Learning 1 (1), 81-106. Kluwer Academic Publishers.

Quinlan, J.R. (1990) Learning logical definitions from relations. Machine Learning 5 (3), 239-266. Kluwer Academic Publishers.

Quinlan, J.R., Compton, P.J., Horn, K.A. in Lazarus, L (1986) Inductive knowledge acquisition: A case study. Second Australian Conference on Application of Expert Systems. New South Wales Institute of Technology, Sydney, Australia.

Sammut, C. in Banerji, R.B. (1986) Learning concepts by asking questions. V: Michalski, R.S., Carbonell, J.G. in Mitchell, T.M. (ur.) Machine learning: An artificial intelligence approach (Volume II). Los Altos: Morgan Kaufmann.

Shapiro, E.Y. (1981) An algorithm that infers theories from facts. Seventh Int. Joint Conference on Artificial Intelligence, 446-451. Vancouver, BC: Morgan Kaufmann.

Sutlič, I. (1991) Integrirano okolje za avtomatsko učenje LINUS. Diplomska naloga, Fakulteta za elektrotehniko in računalništvo, Univerza v Ljubljani.

Wnek, J., Sarma, J., Wahab, A.A. in Michalski, R.S. (1990) Comparing learning paradigms via diagrammatic visualization: A case study in single concept learning using symbolic, neural net and genetic algorithm methods. Fifth Int. Symposium on Methodologies for Intelligent Systems. Knoxville, TN.

MOŽNOSTI ZA VODENJE SISTEMOV S POMOČJO METOD UMETNE INTELIGENCE

INFORMATICA 4/91

Keywords: control of dynamic systems
artifical intelligence, qualitative modelling

Tanja Urbančič
Institut Jožef Stefan

Pri vodenju dinamičnih sistemov se srečujemo s problemi tako pri načrtovanju kot pri nadzorovanju vodenja. Članek utemeljuje potrebo po alternativnih pristopih k reševanju teh problemov. Opisuje možnosti, ki jih pri tem nudijo metode umetne inteligence, zlasti avtomatsko učenje in kvalitativno modeliranje.

TOWARDS CONTROLLING DYNAMIC SYSTEMS USING AI METHODS

Two kinds of problems arise when a dynamic system is to be controlled: designing control and supervising the controlled system. In the paper, the need of alternative approaches to solving these problems is shown. Perspectives given by artificial intelligence methods are surveyed, especially those of machine learning and qualitative modelling.

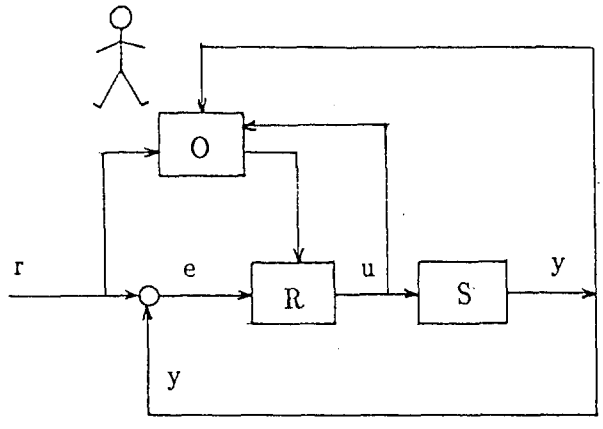
1 Uvod

Vodenje dinamičnih sistemov je zahtevno področje, na katerem se kljub desetletja dolgemu razvoju teorije in akumulaciji izkušenj pojavljajo problemi, ki so s klasičnimi metodami nerešljivi. Pri tem mislimo predvsem na vodenje sistemov, ki jih zaradi njihove zapletenosti ali kompleksnosti ne moremo dovolj natančno predstaviti z matematičnim modelom. Težave nastopijo tudi, če je sprotno zagotavljanje tako natančnih numeričnih vrednosti, kot jih zahtevajo klasične metode, predrago, predolgotrajno ali sploh nemogoče. Namesto čimvečje natančnosti prihaja v ospredje potreba po robustnosti in adaptivnosti, vse bolj pa tudi po razumljivosti vodenja sistemov. Izkazalo se je namreč, da se popolna avtomatizacija nadzora procesa v praksi ne obnese najbolje. Operaterji si želijo predvsem orodje, ki opravlja rutinske operacije in reagira v najkrajšem možnem času, kadar je treba zagotoviti varnost, sicer pa prepušča končno odločitev človeku, pri čemer mu pomaga z interpretacijo opažanj in s predlogi za ukrepanje. Proučiti želimo, kako je na tem področju možno izkoristiti metode umetne inteligence, predvsem metode avtomatskega učenja (Bratko 1989) in kvalitativnega modeliranja (Weld & de Kleer 1990).

2 Identifikacija problemov

Za lažjo identifikacijo problemov si najprej oglejmo grobo poenostavljen prikaz klasično vodenega sistema (slika 1). Sistem (S), ki ga želimo voditi, je skupaj z regulatorjem (R) zaprt v zanko. V tej regulacijski zanki se na podlagi reference (r) in dejanskega odziva (y) računa napaka (e), nato pa se določa in izvaja kontrolna akcija (u). Obstaja več vrst klasičnih regulatorjev, med njimi npr. preprost in široko uporaben PID-regulator, ki upošteva trenutno napako (P-člen), integral napake (I-člen) in predikcijo na osnovi odvoda napake (D-člen):

$$u(t) = k_P e(t) + k_I \int_0^t e(t)dt + k_D e'(t)$$



Slika 1: Regulacijska zanka z operaterjevim nadzorom

Sistem nadzoruje operater. Pogosto se dogaja, da zaradi neustrezne izbire oziroma nastavitve regulatorja ali pa zaradi spremenjenih okoliščin delovanja vodenje sistema ni zadovoljivo. V takih primerih je treba ukrepati. Obstajajo zapletene matematične metode za doseganje robustnosti in adaptacijo, kljub temu pa se zaenkrat pojmuje, da je pri nadzoru in ukrepanju človek nepogrešljiv. Znano je, da je človek sposoben "ročno" voditi tudi dokaj zapletene sisteme. To je možno zaradi njegove sposobnosti *kvalitativnega sklepanja.* Pri tem je bistvena zmožnost opazovanja in interpretacije pojavov v kvalitativnem smislu, pogosto pa ima velik pomen tudi *upoštevanje izkušenj.* Vendar pa je treba upoštevati, da je delo operaterja podvrženo psihofizičnim faktorjem, kot so nemotiviranost, utrujenost ali stres zaradi kritičnosti situacije. Problem je lahko tudi kratek čas, v katerem je treba interpretirati veliko število hkratnih informacij in se odločiti za ukrepanje. Zato je pri tem delu pomoč računalnika lahko zelo dragocena.

Ocenjujemo, da lahko metode umetne inteligence pomagajo reševati probleme pri ključnih fazah, potrebnih za vodenje sistemov:

- pri *načrtovanju klasičnega vodenja* (npr. Pang 1989), zlasti pri določanju numeričnih parametrov regulatorja in morda celo pri sintezi pravil za nastavljanje parametrov,
- pri *adaptivnem vodenju* za sprotno prilagajanje parametrov regulatorja spremembam v procesu (Litt 1991),
- pri *načrtovanju kvalitativnega vodenja* (t.j. vodenja na osnovi kvalitativnih informacij o sistemu), kadar ne moremo uporabiti klasičnih metod, ker ne moremo razviti matematičnega modela, ki bi dovolj dobro odražal realnost (npr. Sammut & Michie 1991),
- pri *nadzorovanju* vodenega procesa: s podporo pri ugotavljanju stanj in dinamičnih tendenc sistema, ki bi lahko privedle do nedopustne degradacije delovanja sistema, ter z nasveti za ukrepanje (Dvorak 1987).

3 Kvalitativno vodenje

Ena izmed možnosti za kvalitativno vodenje sistema je uporaba takoimenovanega *površinskega znanja,* ki je vzorčno vodeno in je ponavadi izraženo s preprostimi pravili oblike

če situacija S, **potem** akcija A

Tovrstno znanje je zelo operativno. Vodenje z njim je preprosto in včasih zadošča. Lahko pa se zgodi, da naletimo na težave pri izgradnji in preverjanju takšne baze znanja. Predvideti in eksplicitno navesti je namreč treba vse možne primere, kar je pri kompleksnejših sistemih seveda vprašljivo. Če se sistem znajde v stanju, ki ga nismo predvideli, je program povsem nemočen. Tudi v primerih, ko je vodenje uspešno, pa lahko nudi le šibko razlago.

Problem izgradnje baze znanja rešujemo z metodami za *avtomatsko sintezo znanja.* Znanje za vodenje sistemov je možno sintetizirati na dva v osnovi različna načina. Pri prvem sintetiziramo znanje neposredno iz poskusov s sistemom, tako da se učimo iz njihovih izidov. Bistvene so numerične evaluacije uspešnosti posameznih kontrolnih odločitev, ki vplivajo na morebitno spremembo kontrolnih odločitev v nadaljevanju poskusov. Na tem principu delujejo znani programi za avtomatsko učenje vodenja sistemov, kot so BOXES (Michie & Chambers 1968), AHC (Barto in sod. 1983; Selfridge in sod. 1985) in CART (Connel & Utgoff 1987). Kot ugotavljamo v članku (Urbančič 1990), so prednosti teh pristopov v tem, da ne potrebujejo poznavanja dinamike sistema in da so zmožni adaptacije

tudi v precej spremenjenih pogojih delovanja. Obravnavali smo tudi njihove pomanjkljivosti, kot npr. netransparentnost zaradi ad hoc izbranih numeričnih parametrov, (pre)dolgotrajno eksperimentiranje in nezanesljivost naučenih pravil. Tem težavam se skušamo izogniti z drugim načinom, pri katerem znanje za vodenje izpeljemo s pomočjo znanja o sistemu, ki ga želimo voditi. O tem govori naslednji razdelek.

4 Kvalitativni modeli in vodenje sistemov

Novejši ekspertni sistemi uporabljajo tudi *globoko znanje*, ki je kompaktnejše ter odraža strukturo in globlje principe problemskega področja. V zvezi s tem se postavljata vprašanji, kako globoko znanje predstaviti in kako ga uporabiti za reševanje problemov.

Globoko znanje lahko predstavimo s *kvalitativnimi modeli*. Ti opisujejo delovanje sistemov in naprav na enostaven simboličen način, ki je blizu človekovemu načinu mišljenja. Natančne numerične vrednosti niso potrebne: vse vrednosti spremenljivke, ki dajo kvalitativno isto vedenje sistema, združimo v eno samo kvalitativno vrednost. Spremenljivke so vezane z relacijami in ne z enačbami kot pri klasičnih modelih. Relacije imajo lahko obliko omejitvenih enačb, neenačb ali logičnih izjav. Podrobnejši pregled področja najdemo v knjigi (Weld & de Kleer 1990).

Končni cilj predstavitve znanja je seveda njegova uporaba. Za razliko od matematičnih modelov, pri katerih rešujemo enačbe in računamo vrednosti funkcij v danih točkah, postanejo kvalitativni modeli operativni s *kvalitativnim sklepanjem*. Spremenljivkam določamo kvalitativne vrednosti tako, da je zadoščeno relacijam v modelu.

Kvalitativni modeli določajo relacije med vhodi v sistem, kvalitativnimi stanji sistema in izhodi. Najpogostejši in "najnaravnejši" način sklepanja začne z znanim začetnim stanjem in vhodi v sistem, nato pa v skladu z modelom razvija nadaljnje vedenje sistema. V tem primeru govorimo o *kvalitativni simulaciji* (npr. Kuipers 1986). Z njo je možno neposredno reševati probleme, kot so na primer napovedovanje vedenja sistema, verifikacija načrtov, izpeljava vseh možnih stanj sistema in prehodov med njimi (envisioning).

Reševati želimo tudi drugačne probleme, ki so v nekem smislu obratni prejšnjim: izhajamo iz podanega vedenja sistema in ugotavljamo, kako lahko do njega pride. Kadar gre za diagnostiko, izhajamo iz manifestiranega nezaželenega vedenja in iščemo stanja, ki so ga povzročila. Pri problemu vodenja sistema je podano zaželeno vedenje, poiskati pa je treba zaporedje vhodov (akcij), ki pri danem začetnem stanju tako vedenje zagotovijo. Za neposredno reševanje takih problemov z modelom je potrebno *vzvratno sklepanje*.

Slika 2: Avtomatska sinteza pravil za vodenje, pristop z izčrpno kvalitativno simulacijo

Učinkovitost sklepanja pri tovrstnih problemih lahko povečamo na različne načine. Eden od njih je modeliranje na različnih nivojih podrobnosti in upoštevanje hierarhij pri sklepanju (Mozetič in sod. 1990): rezultati na manj podrobnem nivoju usmerjajo sklepanje na bolj podrobnem. Druga možnost je uporaba izčrpne simulacije za generiranje enostavnih površinskih pravil, s katerimi je možno hitro reševati probleme na nivoju natančnosti modela. Postopek je ponazorjen na sliki 2 in smo ga že uporabili za avtomatsko sintezo pravil za vodenje inverznega nihala (Urbančič 1991). Če je problemsko področje preobsežno za izčrpno simulacijo in iz ogromne množice možnih primerov velika večina dejansko nikoli ne nastopi, je smiselno uporabiti kombinacijo površinskega in globokega znanja. Za pogoste situacije vgradimo plitvo operativno

znanje, v izjemnih situacijah pa izpeljemo rešitev neposredno iz modela in tako dobljeno pravilo naknadno vključimo v plitvo bazo znanja. Seveda se moramo pred uporabo te metode prepričati, da kritičnost in hitrost procesov to dopuščata.

Medtem ko so se kvalitativni modeli na nekaterih tehničnih področjih kmalu izkazali kot primerno orodje za reševanje problemov, npr. pri diagnostiki (Davis 1984; Genesereth 1984), so možnosti njihove uporabe na področju vodenja sistemov še relativno neraziskane (Franklin 1990). Nekateri rezultati pa vendarle tudi tukaj kažejo na obetavnost kvalitativnega pristopa:

- kvalitativne modele je možno uporabljati za načrtovanje vodenja (npr. Leitch & Francis 1986; Bratko 1991; Urbančič 1991),
- v posebnih primerih (linearni modeli) je možno na podlagi kvalitativnega modela izvesti analizo perturbacij in določiti, kako s perturbacijo vhodnih parametrov dosežemo željeno vedenje sistema (de Mori & Prager 1989),
- kvalitativni modeli lahko pripomorejo k inteligentnemu monitoringu, zaenkrat predvsem v kontekstu diagnostike (Forbus 1987; Dvorak & Kuipers 1989),
- rezultate kvalitativne simulacije lahko uporabimo za analizo stabilnosti vodenih sistemov (Fouche & Kuipers 1991).

Obetavne so tudi študije primerov (npr. Bratko in sod. 1989; Bratko in sod. 1991; Varšek 1991), ki kažejo, da lahko avtomatsko učenje učinkovito uporabimo pri avtomatizirani gradnji kvalitativnih modelov iz opazovanega vedenja sistema.

Vendarle pa še ni bila dosežena stopnja širše praktične uporabnosti. Eden izmed razlogov je gotovo težek problem prevajanja kvantitativnih opažanj v kvalitativni jezik ter kvalitativnih zaključkov v kvantitativne vrednosti kontrolnih akcij. Razen tega je treba poskrbeti za delovanje v realnem času, za upoštevanje različnih tipov znanja (npr. vzročno znanje o sistemu, nujne akcije za zagotavljanje varnosti, verjetnost za posamezne okvare v sistemu in na senzorjih) ter za obvladovanje kompleksnosti. Niso namreč redki sistemi z nekaj tisoč komponentami.

5 Zaključek

Poudariti želimo, da pri razvoju alternativnih metod za vodenje sistemov ne gre za tekmovanje s klasičnimi, kvantitativno orientiranimi pristopi, pač pa za smiselno dopolnjevanje:

- Če je možno sistem v sprejemljivem času res ustrezno predstaviti z matematičnim modelom in so izpolnjene predpostavke za klasične postopke načrtovanja vodenja, bo tako dobljeno vodenje seveda zmeraj natančnejše od kakršnegakoli kvalitativnega pristopa. Veliko prednost predstavljajo tudi teoretično podprte metode za obravnavanje pomembnih vprašanj, kot npr. določanje območja kontrolabilnosti, stabilnosti ipd., ki jih je v takih primerih možno uporabiti. Kvalitativne pristope je v tem kontekstu smiselno uporabiti le kot dopolnilo zaradi zagotavljanja razlage oziroma transparentnosti vodenja ter zaradi nudenja pomoči operaterju pri nadzorovanju procesa.
- Če ustreznega matematičnega modela v sprejemljivem času ne moremo razviti ali pa le-ta (od samega začetka ali pa čez čas, ko se razmere spremenijo) ne odraža dovolj verno realnosti, je treba iskati kompromis med natančnostjo rešitve in hitrostjo oziroma ceno, ki je za to potrebna. Problem bi lahko formulirali kot iskanje konceptualno jasnih, čeprav manj natančnih rešitev, ki bi se obnesle v praksi ne le zaradi razumljivosti, pač pa nenazadnje tudi zaradi nižje cene.

Literatura

Barto, A.G., Sutton, R.S., Anderson, C.W. (1983) Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems. IEEE Transactions on Systems, Man and Cybernetics, Vol. SMC-13, No.5, 834-846.

Bratko, I. (1989) Machine Learning. V knjigi K.J. Gilhooly (ur.) Human and Machine Problem Solving, Plenum Press, New York.

Bratko, I., Mozetič, I., Lavrač, N. (1989) KARDIO: A study in deep and qualitative knowledge for expert systems. Cambridge, MA: MIT Press.

Bratko, I. (1991) Qualitative Modelling: Learning and Control. Sixth Czechoslovak Conference on

Artificial Intelligence, Prague, June 1991.

Bratko, I., Muggleton, S., Varšek, A. (1991) Learning qualitative models of dynamic systems. Inductive Logic Programming Workshop 91, Viana de Castello, Portugal, March 1991. A short version also in Proc. Machine Learning 91, Morgan Kaufmann.

Connel, M.E., Utgoff, P.E. (1987) Learning to Control a Dynamic Physical System. In Proceedings of the 6th National Conference on Artificial Intelligence, Morgan Kaufmann, 456-459.

Davis, R. (1984) Diagnostic Reasoning Based on Structure and Behavior. Special Volume on Qualitative Reasoning about Physical Systems, Artificial Intelligence, 24(1-3), 347-410.

Dvorak, D.L. (1987) Expert Systems for Monitoring and Control. Technical Report AI 87-55, Artificial Intelligence Laboratory, The University of Texas at Austin.

Dvorak, D., Kuipers, B. (1989) Model-Based Monitoring of Dynamic Systems. In Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, August 20-25, Detroit, MI, 1238-1243.

Forbus, K.D. (1987) Interpreting Observations of Physical Systems. IEEE Transactions on Systems, Man and Cybernetics, Vol. SMC-17, No.3, 350-359.

Fouche, P., Kuipers, B. (1991) Introducing Energy into Qualitative Simulation. First European Workshop on Qualitative Reasoning about Physical Systems, Genova, January 1991.

Franklin, J.A. (1990) What is Qualitative Reasoning, and Can We Use it for Control? Proceedings of the 29th IEEE Conference on Decision and Control, Dec. 1990, Honolulu, HI.

Genesereth, M.R. (1984) The Use of Design Descriptions in Automated Diagnosis. Special Volume on Qualitative Reasoning about Physical Systems, Artificial Intelligence, 24(1-3), 411-436.

Kuipers, B. (1986) Qualitative Simulation. Artificial Intelligence 29, 289-338.

Leich, R., Francis, J.C. (1986) Towards Intelligent Control Systems. In Mamdani, A., Efstathiou, J. (eds.) Expert Systems and Optimisation in Process Control. Gower Technical Press, Aldershot, England, 62-73.

Litt, J. (1991) An Expert System to Perform On-Line Controller Tuning. IEEE Control Systems Magazine, Vol.11, No.3, 18-23.

Michie, D., Chambers, R.A. (1968) BOXES: An experiment in adaptive control. In Dale, E., Michie, D. (eds.) Machine Intelligence 2, Edinburgh University Press, 137-152.

de Mori, R., Prager, R. (1989) Perturbation Analysis with Qualitative Models. In Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, August 20-25, Detroit, MI, 1180-1186.

Mozetič, I., Bratko, I., Urbančič, T. (1991) Varying level of abstraction in qualitative modelling. V knjigi J.E.Hayes, D.Michie, E.Tyugu (eds.) Machine Intelligence 12, Oxford University Press, 259-280.

Pang, G. (1989) Knowledge engineering in the computer-aided design of control systems. Expert Systems, Vol.6, No.4, 250-262.

Sammut, C., MIchie, D. (1991) Controlling a "Black Box" Simulation of a Space Craft. AI Magazine, Vol.12, No.1, 56-63.

Selfridge, O.G., Sutton, R.S., Barto, A.G. (1985) Training and Tracking in Robotics. In Proceedings of the 9th International Conference on Artificial Intelligence. Morgan Kaufmann, 670-672.

Urbančič, T. (1990) Avtomatsko učenje vodenja dinamičnih sistemov. Informatica, letnik 14, št. 4, 39-48.

Urbančič, T. (1991) Automatic model-based synthesis of control rules for inverted pendulum. First European Workshop on Qualitative Reasoning about Physical Systems, Genova, January 1991.

Varšek, A. (1991) Qualitative model evolution. International Joint Conference on Artificial Intelligence, Sydney, September 1991.

Weld, D.S., de Kleer, J. (eds.) (1990) Readings in Qualitative Reasoning about Physical Systems. Morgan Kaufmann, San Mateo, CA.

POVEZAVA OSEBNEGA RAČUNALNIKA Z ROBOTSKIM KRMILNIKOM ASEA-IRB

INFORMATICA 4/91

Keywords: computer link, robot controller, personal computer, application protocol

Michele Leonardi, Aleš Klofutar, Ivo Gorkič
Institut Jožef Stefan

Prispevek opisuje povezavo robotskega krmilnika ASEA IRB6/L z osebnim računalnikom. Komunikacija temelji na protokolu asinhronske serijske komunikacije (ADLP10) in aplikacijskem protokolu (ARAP). Prvi del opisuje električno povezavo robotskega krmilnika in osebnega računalnika ter oba protokola. Večji poudarek je na aplikacijskem protokolu, saj ta omogoča dodatke oziroma popravke in dodelave programa brez prevelikih težav. Na podlagi aplikacijskega protokola je nastala knjižnica ukazov. Drugi del vsebuje spisek ukazov in njihovo uporabo ter opisuje delovanje programa `Proto`, ki je nastal hkrati s knjižnico ukazov.

COMPUTER LINK FOR ASEA ROBOT CONTROLLER

The paper presents the computer link for ASEA IRB robot controler. Communication is based on asinchronous communication protocol (ADLP10) and application protocol (ARAP). Electrical link between the controller and the personal computer together with both protocols are presented in the first part of the paper. The application protocol is more emphasized due to its better possibilities of improvement. A library of instructions was developed on the basis of the application protocol. The list of instructions and the description of program `Proto`, which was developed together with the library are described in the second part of the paper.

1. Uvod

Vse večja robotizacija proizvodnih procesov pogojuje potrebo po usklajenem delovanju robotov in njihovem prilagajanju okolici. Robotski krmilniki poskrbijo, da robot opravi vnaprej zadane naloge, katerih obsežnost je omejena z zmogljivostmi robotskega krmilnika. Zmogljivosti robotskega krmilnika lahko bistveno povečamo s povezavo z zmogljivejšim osebnim računalnikom.

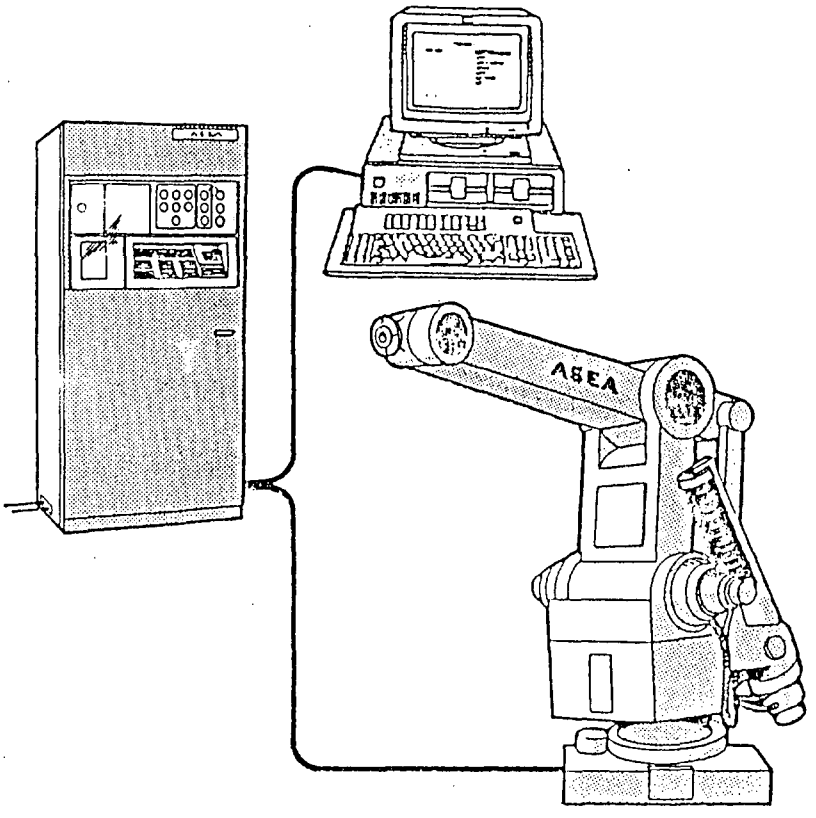
V članku smo opisali povezavo robotskega krmilnika ASEA IRB6/L z računalnikom PC AT. Povezava omogoča izvajanje večine ukazov, ki jih lahko izvedemo neposredno z robotskega krmilnika tudi z osebnega računalnika.

2. Komunikacijski protokol

Povezava med robotskim krmilnikom in osebnim računalnikom je omogočena s protokoloma, ki jih je predpisala ASEA: ADLP 10 (ASEA Data Link Protocol) [2] in ARAP (ABB Robot Application Protocol) [1].

2.1 Električna povezava

Na strani robotskega krmilnika smo uporabili poseben komunikacijski vmesnik DSCA114 in pri njem ustrezno nastavili vhodno/izhodni naslov vmesnika in prekinitveni nivo. Na strani osebnega računalnika pa smo uporabili že vgrajeni vhodno/izhodni vmesnik z ustrezno nastavitvijo.



Slika 1: Povezava robotskega krmilnika z osebnim računalnikom

2.2 ADLP10

ADLP-10 je postopek za asinhronsko komunikacijo med dvema postajama v hierarhičnem sistemu. Postopek temelji na protokolih ECMA-16 in ECMA-24.

Prenos podatkov je asinhronski, serijski in poldvosmeren. Lahko je iniciiran z obeh strani. Vsak znak (podatkovni, kontrolni) je sestavljen iz 8 bitov + parnostni bit + stop bit. Preverjanje pravilnosti komunikacije se vrši z vertikalno in horizontalno kontrolo parnosti.

Za komunikacijo prek serijskega vhodno/izhodnega vmesnika smo razvili podprogram, ki omogoča odpiranje in zapiranje komunikacijskega kanala ter oddajo in sprejem posameznih znakov. Pri odpiranju komunikacijskega kanala je potrebno specificirati številko komunikacijskega kanala (COM1, COM2), hitrost prenosa (npr. 9600 b/s), število bitov na znak, število stop bitov in parnost (npr. liha parnost). Podprogram poleg tega omogoča hkraten sprejem in oddajo znakov ter detektira napake med sprejemom.

Izmenjavo informacij med postajama razdelimo v tri faze

1. Vzpostavitev komunikacije

2. Prenos informacij

3. Zaključek komunikacije

Prenos informacij poteka med drugo fazo. Za spremembo smeri komunikacije moramo izmenjavo zaključiti in ponovno vzpostaviti komunikacijo.

Postajo, ki vzpostavi komunikacijo imenujemo master, podrejeno pa slave. Master je zadolžen za prehajanje med fazami in vzdrževanje komunikacije.

2.3 ARAP

Komunikacija med postajama poteka prek sporočil, ki so definirana z aplikacijskim protokolom (ARAP). Sporočilo predstavlja informacije, ki so med seboj logično povezane in sestavljajo enoto. Sestavljeno je iz enega ali več telegramov.

2.3.1 Telegrami

Informacijo, ki se izmenjuje med robotskim krmilnikom in osebnim računalnikom, razdelimo v manjše enote, ki jim pravimo telegrami. Vsak telegram je sestavljen iz glave in podatkovnega polja. Glava je vedno enake velikosti (8 bytov) in vsebuje informacijo, ki temelji na aplikacijskem protokolu (velikost telegrama, funkcijsko kodo, tip telegrama, indikator dolžine telegrama, status odgovora, naslov pošiljatelja in naslov prejemnika telegrama).

Funkcijska koda določa namen sporočila. V dokumentaciji je predvidenih 44 različnih funkcijskih kod.

Imamo dva tipa podatkovnih polj (koda napake, podatkovno polje). Velikost in vsebina podatkovnega polja se spreminjata v odvisnosti od tipa sporočila in je v določenih primerih tudi izpuščena.

Obstajajo trije različni tipi telegramov:

Ukaz lahko pošljeta tako robotski krmilnik kot osebni računalnik in pomeni, da pošiljatelj zahteva, da sprejemnik izvrši neko nalogo. Ukaz se lahko sestoji le iz glave ali pa iz glave in podatkovnega polja. Podatkovno polje je lahko daljše od maksimalne dolžine telegrama, zato se za ukaz lahko uporabi več povezanih telegramov. Ukaz vedno pričakuje odgovor (pozitiven oz. negativen).

Odgovor je poslan le kot potrdilo ukaza in ga lahko pošljeta tako robotski krmilnik kot osebni računalnik. Tako kot ukaz se tudi odgovor lahko sestoji le iz glave ali pa iz glave in podatkovnega polja. Če je podatkovno polje daljše od maksimalne dolžine telegrama, se za odgovor lahko uporabi več med seboj povezanih telegramov. Lahko ga uporabimo tako za pozitivno kot tudi negativno podrditev ukaza. V primeru negativne potrditve se podatkovno polje nadomesti s 16 bitno kodo napake. Če za neko funkcijsko kodo podatkovno polje ne obstaja, se telegramu doda 16 bitna koda napake v primeru negativnega odgovora.

Spontano sporočilo lahko pošlje le robotski krmilnik računalniku in ni pogojeno z ukazom. Robotski krmilnik pošilja spontana sporočila kot posledico različnih dogodkov v delovanju robota npr. izklop v sili, sistemske napake itd.

3. Knjižnica ukazov

Na osnovi aplikacijskega protokola smo v programskem jeziku Turbo Pascal razvili knjižnico ukazov, ki omogočajo vodenje in programiranje robotskega krmilnika z osebnega računalnika. Osnovo predstavljajo ukazi, ki so z določenimi funkcijskimi kodami direktno dosegljivi iz aplikacijskega protokola.

Vsi ukazi so oblike `Exec_...` in kot prvi parameter vedno vrnejo kodo napake (`ErrResult`), ki jo javi robotski krmilnik. Če je vrednost `ErrResult = 0`, pomeni, da je prišlo do prekinitve komunikacije, če pa `ErrResult = -1`, pa da je robotski krmilnik ukaz izvršil. Ostale vrednosti kode napake pa najdemo v dokumentaciji.

Ukaze delimo na osem skupin.

Statusni ukazi omogočajo branje pozicije in orientacije robota ter branje stanja motorjev robota.

Ukazi povezani z robotskimi programi omogočajo prenos programov z računalnika na robotski krmilnik, brisanje programov iz spomina krmilnika, zagon in ustavitev programov iz krmilnika.

Ukazi povezani s premikanjem robota Za premik robota moramo poslati tri ukaze

1. Začetna točka
2. Ukaz za absolutni ali relativni premik
3. Končna točka

Ukaz omogoča nastavitev hitrosti robota in absolutno oziroma relativno premikanje robota.

Ukazi povezani s koordinatami orodja omogočajo vpis , brisanje in branje koordinat orodja ter aktiviranje določenih koordinat.

Ukazi povezani z registri omogočajo vpisovanje in branje iz

- registrov
- lokacijskih registrov
- digitalnih izhodov

- digitalnih vhodov
- frame registrov

Ukazi povezani s konfiguracijo omogočajo branje in vpisovanje konfiguracije na robotski krmilnik.

Ukazi povezani z Vision sistemom omogočajo branje podatkov, ki so povezani z robotskim vidom s krmilnika in lociranje modelov, ki so v vidnem polju kamere. seveda jih lahko uporabljamo le, če je krmilnik opremljen z Vision sistemom.

Ukazi povezani s prijemalko robota omogočajo zapiranje in odpiranje prijemalke robota.

4. Program Proto

Na podlagi knjižnice ukazov smo razvili program Proto, in z njim preverili delovanje knjižnice ukazov. Poleg osnovnih ukazov, ki smo jih predstavili v prejšnjem poglavju, vsebuje program še nekaj kombinacij le teh, zraven pa še omogoča sprejem ukazov s strani robotskega krmilnika in sprejem spontanih sporočil (Spontaneous messages) s strani robotskega krmilnika.

4.1 Podatki o stanju robota

Program periodično zahteva od robotskega krmilnika podatke o stanju, v katerem se robot nahaja. Zgornja vrstica je rezervirana za sporočila uporabniku. Tu se izpisujejo naslednje stvari

- Stanje pripravljenosti
- Vsebina spontanih sporočil [1, str.78]
- V primeru nepravilno sprejetega ukaza s strani robotskega krmilnika, sporočilo, ki ga definira koda napake [1, str.81]
- Potrditev uspešno izvedenega ukaza.

V drugi vrstici se izpisuje stanje motorjev robota (Stand by, Operate, Execute, Emergency stop) ter program in vrstica programa, v kateri smo. V tretji vrstici imamo stanje enote za programiranje (connected, disconnected). V četrti pa pozicija (v mm) in orientacija (v stopinjah) robota.

Spodnji dve tretjini ekrana sta rezervirani za ukaze uporabnika.

4.2 Ukazi

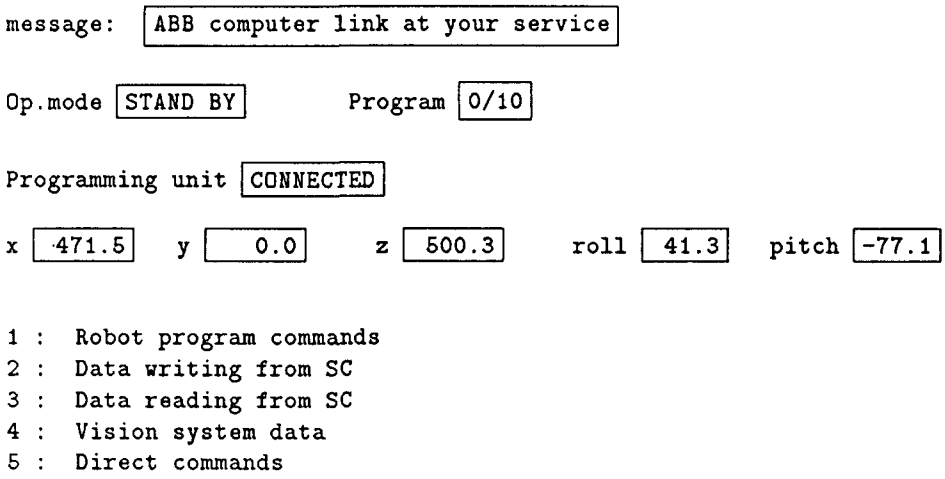
Ukazi so razdeljeni na pet sklopov

1. Ukazi povezani s programiranjem robotskega krmilnika (`Robot program commands`)
   Uporabljajo ukaze povezane z robotskimi programi iz knjižnice ukazov poleg tega pa še izpis programov na ekran in tiskalnik.
2. Branje podatkov z robotskega krmilnika (`Data reading from SC`)
   Omogočajo branje podatkov iz posameznih registrov robotskega krmilnika, branje konfiguracije in koordinat orodja.
3. Vpisovanje podatkov na robotski krmilnik (`Data writing from SC`)
   Omogočajo vpis podatkov iz posameznih registrov robotskega krmilnika, vpis konfiguracije in koordinat orodja.
4. Ukazi povezani z robotskim vidom (`Vision system data`)
   Uporabljajo ukaze povezane z robotskim vidom iz knjižnice ukazov.
5. Neposredni ukazi (`Direct commands`)
   Omogočajo aktiviranje določenih koordinat orodja in določenega koordinatnega sistema odpiranje in zapiranje prijemalke, sinhronizacijo, vžig in izklop motorjev robota ter premikanje robota. Premikanje robota je mogoče
   - absolutno ali relativno: podamo koordinate pozicije (v mm) in orientacije (v stopinjah) robota.
   - zvezno: s pritiskom na določeno tipko se robot premakne v smeri, ki jo določa tipka.

5. Zaključek

Programski paket omogoča povezavo robotskega krmilnika ASEA z osebnim računalnikom.

Trenutno stanje robota in njegove okolice lahko izčrpnejše in hkrati bolj pregledno prikažemo na



Slika 2: Program Proto

zaslonu osebnega računalnika kot na dvovrstičnem prikazovalniku robotove prenosne ukazne plošče. S tem je mogoče statistično spremljanje in iskanje optimalnih časov delovnih faz.

Pomembna je tudi možnost pisanja programov za robotski krmilnik na osebnem računalniku. S tem prenesemo programiranje iz proizvodnih dvoran v človeku prijaznejše okolje in omogočimo simulacijo delovanja robota na osebnem računalniku. Omogočena nam je tudi uporaba vseh programskih orodij namenjenih za izdelavo programov, kar nam v veliki meri olajša pisanje le teh.

Ob vseh drugih prednostih je zelo pomembna tudi možnost implementacije nove programske opreme. Robotski krmilnik ASEA namreč predstavlja zaprt sistem in implementacija nove programske opreme je mogoča le prek povezave z osebnim računalnikom.

Programski paket je zgrajen modularno in omogoča lahko vključevanje knjižnice ukazov v programe napisane v programskem jeziku Turbo Pascal. Omogočena je kontrola napak, saj rutine kot enega izhodov vračajo kodo napake.

Program `Proto`, ki smo ga razvili za preizkus delovanja knjižnice ukazov poleg osnovnih funkcij omogoča še izpis robotskih programov na ekran in tiskalnik.

Programski paket bi lahko uporabili tudi za uskladitev delovanja več robotov. Protokol to omogoča, seveda pa so potrebne tudi ustrezne dodelave programskega paketa in pa algoritmi, ki bi koordinirali sodelovanje več robotov.

Literatura

[1] ABB Robot Application Protocol (ARAP) for Computer Link ABB, Sept. 1988

[2] ASEA Asynchronous Communication for Computer Link-ADLP 10, Jan. 1984

[3] ABB Industrial Robot System IRB6/2, July 1983

[4] Ivo Gorkič: Program za povezavo industrijskega robota z osebnim računalnikom, Diplomska naloga, Fakulteta za elektrotehniko in računalništvo, Ljubljana, Junij 1991

Novice in zanimivosti

64 let dvoma o nastajanju umetne inteligence

(Kritika umetne inteligence Huberta L. Dreyfusa)

Anton P. Železnikar

V zgodovini kritike umetne inteligence bo H.L. Dreyfusu lahko da pripadalo podobno mesto, kot ga ima Sokrat v zgodovini zapadne filozofije. Kot pravi sam [MOM, str. 202], *»za Nietzscheja Sokrat ni bil junak naše kulture, temveč njen prvi izprijenec [degeneriranec], ker je zgubil zmožnost aristokracije, da zaupa intuiciji. „Možje poštenja ne razgaljajo svojega razuma na ta način," je pripomnil Nietzsche.«* Dreyfusova vloga v evforičnem in nekritičnem pohodu umetne inteligence v šestdesetih in sedemdesetih letih tega stoletja je bila v prikazovanju naivnosti zaupanja v neko tehnološko (računalniško) intuicijo (ideologijo), ki je napovedovala skorajšnje doseganje in preseganje človekove inteligence (uma) s pomočjo matematično-teorijskega simbolizma in strojev, ki ta logični aparat oziroma logično strojništvo tehnološko podpirajo.

Čeprav je bil ameriški filozof H.L. Dreyfus, sicer učenec Dagfina Foellesdala, znanega interpreta fenomenološke koncepcije Edmunda Husserla, prvi radikalni kritik zamisli in ideologije umetne inteligence, ki je začela nastajati v začetku šestdesetih let na M.I.T., pa se je kritika tega, kar se je v petdesetih poimenovalo umetna inteligenca, pojavila že prej. Dejansko se kritika mišljenja inteligence (razumevanja) kot človekovega pojava mišljenja pojavlja tako ali drugače in v vsej domišljeni zapletenosti in eksplicitno v delu Martina Heideggra, *Sein und Zeit,* ki izide v letu 1927. Dvom o tem, o čemer so še lahko upali učenci Husserla, dobi s Heideggrovim delom spoznavno ozadje, ki ga zlagoma, kasneje pa čedalje bolj intenzivno prevzamejo tudi mlajše generacije iz komune ameriške umetne inteligence in t.i. računalniških znanosti po svetu.

Kritike umetne inteligence seveda ni mogoče ločiti od kulturnih paradigem modernizma in postmodernizma. Gre za določeno kontinuiteto razvoja znanosti, ki je še pretežno zasidrana v moderni, njena kritika pa se že razteza v t.i. postmodernost. Zgleden opis pojava postmodernosti, ki ni le muha enodnevnica, temveč postaja temeljno metafizično vprašanje človeka sedanjosti, je Slovencem podaril A. Debeljak v *Postmoderni sfingi* [PS]; ob tem se odpira (postmodernistično) vprašanje, ali je to delo le po naključju izšlo v Avstriji. Umetna inteligenca (UI) je kot znastvena in tehnološka disciplina danes še vedno utemeljena v matematičnih in računalniških programskih sistemih, kot posebni (»inteligenčni«) del algoritmike, ki obvladuje včeraj in še danes metodologijo in konceptualizem računalniške znanosti. Pri tem je *computer science* ustrezni termin za raziskovalno in pedagoško dejavnost na ameriških univerzah in institutih. Opravičilo algoritmike, ki je sinonim za dobro (definitivno) opredeljenost matematičnih izrazov, programov in podatkov, je utemeljeno s posebnim podatkovnim orodjem, ki se imenuje računalnik, tj. stroj za procesiranje programskih kodov, ki so lahko aktivni in pasivni simboli, tj. procedure in pripadajoči podatki. Kritika UI pa se napaja v filozofiji, ki je hkrati kritika kartezijanskega horizonta, značilnega za ustroj znanosti v modernizmu.

Profesor H.L. Dreyfus je 10. junija 1991 predaval na Univerzi v Gradcu (Avstrija) [Potrč, HUI] o *Heideggrovi kritiki umetne inteligence.* Heidegger kritizira več stvari hkrati: Husserlovo razumevanje intencionalnosti, kibernetiko in primerjavo človeškega bitja z računalnikom. Simbolni sistemi naravoslovnih znanosti se praviloma utemeljujejo z reprezentacijskimi (predstavitvenimi) procesi (posebnimi jeziki samimi ali govoricami samimi), kjer se pojavlja problem, kako pojasnjevati razmerja simbolov in stvari v svetu; ta problem se danes pojavlja tudi v okviru

simbolnih sistemov, ki opisujejo modele uma (duha). Gre za tri stališča, ki so vredna pozornosti: (1) Temeljno razmerje bitja s svetom je intencionalnost [*pripomba avtorja:* npr. neke vrste biološka in trenutna mentalna vztrajnost oziroma vztrajanje], ki ni predstavitvena (reprezentacijska), je brez predstavitvene vsebine. (2) To, kar oblikuje enoto, so bistva v svetu in naše spretnosti (veščine) v rokovanju z bistvi. To je izhodišče *praktičnega holizma*, ki je v nasprotju s teorijskim holizmom. (3) Simboli imajo pomen le v okviru vsakdanjih pomenskih dejavnosti. Očitno je, da je stališče (3) izpeljano iz stališč (1) in (2).

Heidegger pravi, da imajo simboli pomen (zgolj) zaradi naših dejavnosti, ki so povezane z veščinami. Nasprotuje predikativnim pomenom oziroma simbolom v Husserlovem pomenu, kjer smo v razmerju (referiranju) s svetom zaradi predikativnih pomenov. Heidegger nasprotuje tako vzročnim relacijam (npr. if-then-izem v matematiki, in zlasti v računalniških, umetno-inteligenčnih in programskih strukturah), kot tudi razlikovanju internalizma in eksternalizma. Heideggrova kritika je tedaj usmerjena na predstavitveno, reprezentacijsko intencionalnost, intencionalnost sama pa je po Dreyfusu izredno pomembna v Heideggrovem načinu mišljenja. Preobrat, ki ga Heidegger predlaga po tradiciji, dolgi dva tisoč let, je v prednosti prvobitne intencionalnosti, katere bistvo je v dejavnosti, ki temelji na veščinah. Husserlovska intencionalnost je izjemna; pojavlja se le tedaj, ko določena dejavnost ni uspešna, ko določene veščine še ne obvladujemo (npr. v fazi učenja ali v nepredvidenih okoliščinah).

Heigegger očita Husserlu, da predpostavlja nekakšna bistva duha, kritizira tiho uvajanje razlike med subjektom in objektom, med predmetom in svetom, fenomenološko redukcijo, Husserlov internalizem. Intencionalnost je tako v vedenju (obnašanju), ki je biti-usmerje-na. Temeljna intencionalnost je v vedenju in ne v zavesti, je brez metafizične podlage.

Uporaba kladiva za zabijanje je pri Heideggru znani primer fenomena. Pri uspešni uporabi kladiva izgine kladivo-orodje v svoji priročnosti. Pomemben je tedaj namen naše dejavnosti in ne predmet sam. Subjekt, ki zabija s kladivom, je brez prepričanja. Ima veščino, da to opravlja, je vpleten v dogodje zabijanja. Tubit je zaskrbljena vključenost v svet, ko je potrebno vedeti kako, a ne vedeti kaj, kjer ni pomembno propozicionalno znanje. Ni razlikovanja med svetom in menoj. To razlikovanje se pojavi šele pri motnjah, ko rokovanje ne poteka brez težav.

V čem je tedaj Heideggrova kritika UI? V Descartesovem kognitivizmu. Nasproti Descartesovim predstavam oziroma reprezentacijam postavlja Heidegger stvari v uporabi, s katerimi rokujemo. Pri kognitivizmu je fizikalni simbolni sistem realiziran v duševnosti, pri UI pa v računalniku. Kritika se začne z medsebojnim napotovanjem (spoznavanjem) orodja. Kladivo se napotuje na žebelj, žebelj na les, les na gradnjo hiše in hiša na mizarje (if-then-istični predikativni niz). V UI se za veščine in načine rokovanja dodajajo le funkcijski predikati. Tako se osmišljajo izolirana, nepomembna bistva v povezavi z drugimi izoliranimi, nepomembnimi bistvi. Pomembne so le funkcije delov in napotovanje na dele. Descartesov svet je svet atomarnih lastnosti. Šele ko se pojavi problem v funkciji kladiva, stopi kladivo kot predmet v ospredje.

Heidegger in njegov interpret Dreyfus poudarjata [HUI], da rokujejo ljudje v UI z osiromašenimi elementi. UI in logika uvajata izolirane predikate in skozi nje razumevata svet v terminih napovedi (aluzij) in v izrazoslovju predikatnega računa. Ker se pri Descartesu interpretacija, pojasnjevanje sveta začenja z notranjim svetom, lahko rečemo, da UI umanjka pomembnost: nastane vprašanje, kako to pomembnost vzpostaviti. Ko se v računalnik vnaša več in več dejstev (bistev), sistem ne postaja inteligentnejši, temveč se čedalje bolj oddaljuje od pomembnosti. Tudi pravila veščin, ki se vnašajo, so pravila tipa znanje-kaj in ne znanje-kako. Inteligenca veščin pa v UI ne obstaja, saj se same veščine upirajo matematični formalizaciji in programiranju. Optimizem, kot je bil napovedan na začetku UI, ni uspel.

Zanimivo je, da je Dreyfusu medtem uspelo izdati knjigo *Being-in-the-world* pri založbi

M.I.T. Press [BIW]. Zakaj zanimivo? Winograd in Flores [UCC] navajata kot referenco [WIB] že leta 1986. Ali je Dreyfusova najnovejša knjiga čakala na objavo pri M.I.T. Press kar nekaj let (pet let ali več), da se je morda zgladil spor med Dreyfusom in M.I.T. iz šestdesetih let, ko je Dreyfus v svoji ekspertizi o perspektivah UI (ocena je bila napisana za RAND) primerjal UI z alkemijo [AAI]. V tej oceni je Dreyfus med drugim zapisal tole [MOM, str. 8-9]: *»Zgodnji uspehi pri programiranju digitalnih računalnikov, ki kažejo na enostavne oblike inteligentnega vedenja, se povezujejo s prepričanjem, da je razlika glede na inteligentno aktivnost le v stopnji in kompleksnosti; verjame se, da je procesiranje s kognitivno zmožnostjo mogoče formulirati s programom in simulirati na digitalnem računalniku. Poskusi simuliranja kognitivnih procesov pa so naleteli na večje težave, kot se je pričakovalo.*

Raziskava teh težav odkriva, da se pri poskusu analize inteligentnega vedenja v jeziku digitalnega računalnika sistematično zanemarjajo tri osnovne človeške oblike procesiranja (meje zavesti, ločevanje bistvo/naključje in toleranca dvoumnosti). Bistveni razvoj v umetni inteligenci bo moral počakati na računalnike popolnoma drugačne vrste, od katerih so edini obstoječi [možni] prototip slabo razumljeni človekovi možgani.«

Zaradi te svoje ocene je imel Dreyfus na M.I.T. velike težave. Najprej je bila preprečena distribucija kritičnega poročila. Študentje in profesorji, ki so delali na robotskem projektu, se niso smeli družiti z Dreyfusom niti pri kosilu. Profesor J. Weizenbaum, ki je tudi dvomil o verodostojnosti značilnih izjav, se je z Dreyfusom skrivoma sestajal doma. Dreyfusa so bolje razumeli v Novosibirsku, na Japonskem in v Bell Telephone. S svojo kritiko UI se je Dreyfus mimogrede dotaknil tudi filozofije biti v okviru postmodernega tematskega sklopa, ki pomeni vobče detronizacijo racionalnosti, konec kartezijanskega subjekta in prevlado informacije (znanja) nad teoretsko-abstraktnim kompleksom racionalizma. Zaustavljanje Dreyfusove kritike s strani njegovih kolegov na univerzi pa kaže na (trenutno?) pomanjkanje profesionalnega etičnega koda, ki je še značilnost iztekajočega modernizma ne le na M.I.T., temveč tudi drugod po svetu.

V okviru Dreyfusovega predavanja v Gradcu zasledimo v Potrčevem spisu [HUI], ki brškone spet ni bil slučajno objavljen izven Slovenije, še neko izjavo, ki kaže morda na trdovratno značilnost slovenske kulturne (filozofske) tradicije. Čeprav sega poznavanje Heideggrovega dela v Sloveniji še v predvojno obdobje pa pozornost slovenskih heideggerjevcev za filozofijo vobče upada in to pomanjkanje interesa za Heideggra tudi ni včerajšnje, čeprav interes za Heideggra v svetu narašča. Ali gre morda za kakovosten (novi ideološki) premik iz domene filozofije v domeno kulture, ki bi lahko bil značilnost nekega koncepta nastopajoče slovenske Kulturgesellschaft?

Naslednje vprašanje, ki se ponuja samo po sebi, je, zakaj Dreyfusova nova knjiga Being-in-the-world obravnava prav kompleks biti-v-svetu. Kot kritik UI preučuje Dreyfus seveda tudi inteligenco, ki je reagiranje v vedenju bitja, ki se odziva na trenutne okoliščine v svetu, na tiste, ki bitje vznemirjajo ali bi ga lahko vznemirile. V svojem bistvu je tedaj razprava o biti-v-svetu tudi razprava o tem, kar vulgarno (neprofesionalno) imenujemo inteligenca. Inteligenca je le obči termin v vrsti njenih pojavnosti, kot so npr. v filozofskem jeziku Heideggra analiza tubiti, bit-v-svetu, svetovnost sveta, so-bit in samo-bit, v-bit in še skrb kot bit tubiti. To pa je prvi odsek (Erster Abschnitt) Heideggrovega *Dela Sein und Zeit.* V okviru petega poglavja z naslovom *V-bit kot taka* se obravanavata *Tu-bit kot razumevanje* (par. 31) in *Razumevanje in razlaga* (*nem.* die Auslegung, *angl.* interpretation) (par. 32). Ker je že osnovno Heideggrovo delo Sein und Zeit zgodnji program postmodernizma, lahko le upamo, da bo Dreyfusova Being-in-the-world morda postala tudi temelj pri študiju možnosti prihodnjih inteligentnih artefaktov — ne le po svetu, temveč tudi v krogih domače UI. In tako prihajamo naposled do vprašanja, ki zadeva slovensko usmeritev UI in njeno perspektivo v postmoderni epohi.

Osnovno usmerjenost slovenske UI je mogoče izluščiti iz njenih del (objav, uporabnih artefaktov, mednarodne povezanosti in odmevnosti). Ali je ali ni osnovna usmeritev domače UI predvsem sholas-

tična? Pridevnik *sholastičen* seveda ne aludira na pomen srednjeveškega sholasticizma; poudarja pa šolsko, šolniško in vzgojno naravnanost (*angl.* scholastic attainments); nanašajoč se na dovolj pedantno prenašanje in posnemanje (modernističnega) znanja, ki je bistveno in značilno za t.i. sekundarno izobraževanje (*angl.* scholastic meet). Na tem mestu ostaja odprto vprašanje lastnih raziskovalnih pretenzij in povezav s centri, ki oblikujejo postmodernistično paradigmo UI in seveda tudi lastnih izvirnih dosežkov. O tem pa bo lahko povedanega kaj več in bolj konkretno na drugem mestu. Slovenska UI se bržkone ne približuje kritičnosti pogleda Dreyfusa, saj bi bilo zanj potrebno vzpostaviti ne le zahtevnejše, temveč tudi drugačno razumevanje UI.

V okviru postmodernosti nastajajo tudi nova izhodišča pri snovanju t.i. inteligentnih informacijskih sistemov. Zanimivo je, da se pridevnik *inteligenten* čedalje bolj poredko uporablja. Na površje prihaja zavest, da z obstoječimi računalniki ni mogoče simulirati inteligence, lahko pa se uporabi v posameznih primerih vsa razpoložljiva informacija. Gre v bistvu za ad hoc pristop pri razvoju (načrtovanju, programiranju) zahtevnih informacijskih sistemov, kot je npr. japonski projekt elektronskega slovarja, ki vsebuje med drugim operacijo prevajanja stavkov iz enega jezika v drugi jezik. Ta projekt vzpostavlja ne le posebne informacijske standarde, temveč skupaj z njimi, tudi razvojna orodja za doseganje zahtevnih standardov. Študij in izpopolnjevanje teh ad hoc informacijskih sistemov bo lahko opravljen šele po njihovi uspešni uporabi na trgih svetovnih jezikov.

UI je v prejšnjih desetletjih nastajala (in ponekod še nastaja) predvsem kot *Velika zgodba* in manj kot znanstveno konsenzualna disciplina. Določen konsenz je dobivala predvsem v lastni komuni in deloma v komunah s podobno problematiko (npr. v cognitive science). *Toda Velike zgodbe so se zgodovinsko razvrednotile ravno na podlagi spekulativnega prepričanja, da je mogoče določen diskurz opravičiti le, če ga legitimiziramo kot nosilca potencialov* (npr. inteligence, razuma, osvoboditve človeštva) [PS, str. 80].Znanosti, kot najvišje oblike racionalnosti, so začele same spodkopavati pozicijo razuma. Npr. naravoslovne znanosti so se same odrekle prentenziji na absolutno resnico, poudarjajoč, da velja resnica le znotraj njihove posebne paradigme (hermenevtike, logosa, metodologije). Takšen razvoj znanosti je tako že nakazal strukturo nastajajoče postmoderne znanosti (kulture). Tudi redefinirani koncept informacije in informacijskega sistema kot *majhna zgodba* informiranja stvari oblikuje svojo postmoderno paradigmo [IP], ki naj bi razvijala med drugim nove oblike razumevanja informacijskih strojev (orodij, programov) kot artefaktov t.i. postračunalniškega tipa. Ta diskurz, ki se uveljavlja v postmodernem razmerju ob zatonu modernih legitimizacijskih mehanizmov, ima značaj majhne zgodbe, začasnega dogovora, lokalno obliko konsenza in kakor je lahko resnica na eni strani, ni več resnica na drugi [PS, str. 83].

Slovstvo

[PS] Debeljak, A., Postmoderna sfinga, Založba Wieser, Celovec-Salzburg (1989).

[AAI] Dreyfus, H.L., Alchemy and Artificial Intelligence, The RAND Corporation Paper P-3244, December 1965.

[WCD] Dreyfus, H.L., What Computers Can't Do: A Critique of Artificial Reason, Harper & Row, New York (1972).

[MOM] Dreyfus, H.L. and S.E. Dreyfus, Mind over Machine, The Free Press, A Division of Macmillan, Inc., New York (1986).

[BIW] Dreyfus, H.L., Being-in-the-world: A Commentary on Division I of Heidegger's Being and Time, M.I.T. Press, Cambridge (in press since 1985, appeared finally, in 1991).

[HUI] Potrč, M., Heideggerova kritika umjetne inteligencije, Folozofska istraživanja 11 (1991) (40), sv. 1, 91- 97.

[UCC] Winograd, T. and F. Flores, Understanding Computers and Cognition: A New Foundation for Design, Ablex PC, Norwood, New Jersey (1986).

[IP] Železnikar, A.P., Information and Postmodernism, Cybernetica 34 (1991) 67-73.

Avtorsko stvarno kazalo časopisa Informatica, letnik 15 (1991)

Authors Subject Index of the Journal Informatica, Volume 15 (1991)

Članki — Articles

Baar, M., Izvedba in uporaba posebne funkcionalno zasnovane računalniške tipkovnice za mrežo osebnih računalnikov, Informatica 15 (1991), No. 1, 83-87.

Barle, J. and J. Grad, Design, Implementation and Testing of a Primal-dual Interior Point Method for Solving Linear Programs, Informatica 15 (1991), No. 4, 5-11.

Bojadžiev, D., Meaning, Understanding, Self-reference, Informatica 15 (1991), No. 2, 1-5.

Bošnjak, S. and L. Šereš, Case Tools — Software Development and Maintenance Aid: A Survey, Informatica 15 (1991), No. 1, 43- 47.

Bruha, I., Neural Sets: Survey ans Application to Waveform Processing, Informatica 15 (1991), No. 1, 27-42.

Čančer, Vesna, Računalniški podprogrami za računanje lastnih vrednosti in lastnih vektorjev, Informatica 15 (1991), No. 2, 65- 70.

Dedić, A., R. Murn, D. Peček, Digitalizacija in vektorizacija črtnih risb velikih dimenzij, Informatica 15 (1991), No. 4, 60- 68.

Djonova Popova, Iskra, Routing Algorithms for Packet Switched Networks, Informatica 15 (1991), No. 2, 6-13.

Dobnikar, A., Jelena Ficzko, Mira Trebar, Ocenjevano učenje v nevronskih mrežah, Informatica 15 (1991), No. 4, 53-59.

Dujmović, J.J., Primena višedimenzionalnih nizova u programima za merenje računarskih performansi, Informatica 15 (1991), No. 3, 10-14.

Gams, M. and V. Križman, The Principle of Multiple Knowledge, Informatica 15 (1991), No. 2, 23-28.

Karalič, A. and V. Pirnat, Significance Level Based Classification with Multiple Trees, Informatica 15 (1991), No. 1, 54-58.

Kobold, Viktorija and M. Špegel, Monitoring of Real-time Systems, Informatica 15 (1991), No. 1, 1-10.

Kolbezen, P., P. Zaveršek, Hierarhični večprocesorski sistem, Informatica 15 (1991), No. 1, 65-76.

Kononenko, I., Zvezne Bayesove nevronske mreže, Informatica 15 (1991), No. 2, 49-53.

Lakner, Barbara, Določanje položaja teles v prostoru iz ene same 2-D slike, Informatica 15 (1991), No. 1, 77-82.

Lavrač, Nada, Uporaba predznanja v induktivnem učenju, Informatica 15 (1991), No. 4, 69-78.

Leonardi, Michele, A. Klofutar, I. Gorkič, Povezava osebnega računalnika z robotskim krmilnikom ASEA-IRB, Informatica 15 (1991), No. 4, 84-88.

Likar, M., N. Guid, Planiranje s Prologom, Informatica 15 (1991), No. 4, 36-42.

Mernik, M. and V. Žumer, Implementation of Denotational Semantics, Informatica 15 (1991), No. 1, 48-53.

Meško, Tjaša, Pattern Recognition Using Kohonen Maps and Multi- layer Perceptrons, Informatica 15 (1991), No. 4, 12-18.

Mrdalj, S. and V. Jovanović, Structured Object-oriented System Decomposition, Informatica 15 (1991), No. 1, 22-26.

Popov, O.B., Consciousness as a Perequisite for Knowledge, Informatica 15 (1991), No. 3, 1-8.

Popov, O.B., Modes for Reasoning about Consciousness, Informatica 15 (1991), No. 4, 1-4.

Pučko, Marjeta, Analiza zmogljivosti mehanizma za kontrolo pretoka v mrežah za prenos podatkov, Informatica 15 (1991), No. 3, 63-68.

Radovan, M., ER jezik: prijedlog proširenja, Informatica 15 (1991), No. 3, 44-53.

Rajkovič, V., E. Delidžakova-Drenik, B. Urh, Primerjalna analiza treh orodij za izgradnjo in uporabo baze znanja ekspertnega sistema, Infor-

matica 15 (1991), No. 3, 22-28.

Raman, S., L.M. Patnaik, J. Šilc, and M. Špegel, Parallel Implementation of VLSI Circuit Simulation, Informatica 15 (1991), No. 2, 14-22.

Robič, B., Polona Blaznik, O vzporednem množenju matrik, Informatica 15 (1991), No. 1, 59-64.

Trstenjak, Natalija, B. Žalik, N. Guid, Algoritem za pretvorbo telesa predstavljenega z ovojnico v predstavitev z osmiškim drevesom, Informatica 15 (1991), No. 2, 54-64.

Urbančič, Tanja, Možnosti za vodenje sistemov s pomočjo metod umetne inteligence, Informatica 15 (1991), No. 4, 79-83.

Zupan, J., »Slonček«, program za geslenje slovenskega teksta, Informatica 15 (1991), No. 3, 15-21.

Zaveršek, P., P. Kolbezen, Dinamično dodeljevanje procesov na polju transputerjev, Informatica 15 (1991), No. 4, 43-52.

Žalik, B., N. Guid, A. Tibaut, A. Vesel, Algoritem za določitev povezovalnih funkcij krivulj B-zlepkov in NURB krivulj v polinomskem času, Informatica 15 (1991), No. 3, 54-62.

Železnikar, A.P., Informational Models of Dictionaries I, Informatica 15 (1991), No. 1, 11-21.

Železnikar, A.P., Informiranje stvari, Informatica 15 (1991), No. 2, 29-39.

Železnikar, A.P., Informiranje stvari II, Informatica 15 (1991), No. 3, 29-43.

Železnikar, A.P., Formal Informational Principles, Informatica 15 (1991), No. 4, 19-35.

Žerovnik, J., Algoritem ohlajanje, Informatica 15 (1991), No. 2, 40-48.

Knjižne recenzije — Book Reviews

Knjižne novice, Informatica 15 (1991), No. 1, 88-90.

Dujmović, J.J., O knjizi i njenom autoru: Anton P. Železnikar, »On the Way to Information«, Slovensko društvo Informatika, 1990. Informatica 15 (1991), No. 3, 69-70.

Motaln, V., On ther Way to Information (Na poti k informaciji), Informatica 15 (1991), No. 2, 79.

Motaln, V., A.P. Železnikar, Vprašanja (ki jih je postavil dr. V. Motaln avtorju na tiskovni konferenci v Mladinski knigi, 29.1.1991. Informatica 15 (1991), No. 2, 79-82.

Souček, B., Recenzija knjige: Anton P. Železnikar, On the Way to Information, Part 1, Ljubljana. Informatica 15 (1991), No. 2, 78- 79.

Novice in zanimivosti - News

Železnikar, A.P., Longman Dictionaries, Informatica 15 (1991), No. 2, 71-72.

Železnikar, A.P., Projekt japonsko-nemškega slovarja, Informatica 15 (1991), No. 2, 72.

Železnikar, A.P., Konzorcij za leksikalne raziskave, Informatica 15 (1991), No. 2, 73.

Železnikar, A.P., Genelex: Eureka za lingvistično inženirstvo, Informatica 15 (1991), No. 2, 73-74.

Železnikar, A.P., Oblikovanje leksikalnih modelov pri IBM, Informatica 15 (1991), No. 2, 74-75.

Železnikar, A.P., Kitajski elektronski slovar, Informatica 15 (1991), No. 2, 75.

Železnikar, A.P., Slovenski slovnični in besedni pregledovalnik, Informatica 15 (1991), No. 2, 75-76.

Železnikar, A.P., Japonska pobuda za sodelovanje pri razvoju in prodaji elektronskih slovarjev, Informatica 15 (1991), No. 2, 76- 78.

Železnikar, A.P., Dvajsetletnica kongresa IFIP'71, Informatica 15 (1991), No. 3, 70.

Železnikar, A.P., 64 let dvoma o nastajanju umetne inteligence, Informatica 15 (1991), No. 4, 89-92.

Informatica

Informatica *is published four times a year in Winter, Spring, Summer and Autumn by the Slovene Society Informatika, Tržaška cesta 2, 61000 Ljubljana, Yugoslavia.*

Volume 15 Number 4 November 1991 YU ISSN 0350-5596

Informatica

A Journal of Computing and Informatics

C O N T E N T S

Modes for Reasoning about Cosciousness	*O.B. Popov*	1
Design, Implementation and Testing of a Primal-dual Interior Point Method for Solving Linear Programs	*J. Barle* *J. Grad*	5
Pattern Recognition Using Kohonen Maps and Multi-layer Perceptrons	*Tjaša Meško*	12
Formal Informational Principles	*A.P. Železnikar*	19
Planning with Prolog (in Slovene)	*M. Likar* *N. Guid*	36
Process Allocation in the Multitransputer Network (in Slovene)	*P. Zaveršek* *P. Kolbezen*	43
Reinforcement Learning in Neural Networks (in Slovene)	*A. Dobnikar* *Jelena Ficzko* *Mira Trebar*	53
Digitalization and Vectorization of Large Scale Drawings and Maps (in Slovene)	*A. Dedić* *R. Murn* *D. Peček*	60
The Use of Background Knowledge in Inductive Concept Learning (in Slovene)	*Nada Lavrač*	69
Towards Controlling Dynamic Systems Using AI Methods (in Slovene)	*Tanja Urbančič*	79
Computer Link for ASEA Robot Controller (in Slovene)	*Michele Leonardi* *A. Klofutar* *I. Gorkič*	84
NEWS: 64 Years of Doubt of AI (in Slovene)	*A.P. Železnikar*	89
Authors Subject Index of the Journal Informatica, Volume 15 (1991)		93