A Connectionist Knowledge Based System using Coarse-Coded Distributed Representations

S G Sanjeevi
Dept. of Comp. Science & Engineering
N.I.T. Warangal
Warangal, India
91-870-2462711
sgsanjeevi@yahoo.com

P Bhattacharyya
Dept. of Comp. Science & Engineering
IIT Bombay
Mumbai-400076
91-22-5767718
pb@cse.iitb.ac.in

ABSTRACT

Traditionally reasoning systems have been implemented using symbolic methods of artificial intelligence. Connectionist methods of implementing reasoning systems form an alternative paradigm. Among the connectionist reasoning systems two types of representational methods can be used. They are i) localist and ii) distributed representational methods. In the literature, some localist methods for reasoning were used in connectionist systems. Since those systems used localist representations, advantages of distributed representations are not obtainable by them. In this paper, we describe the design and implementation of a connectionist knowledge based system which integrates a connectionist predicate logic reasoning system and a connectionist semantic network. The system uses distributed coarse-coded representations. The connectionist predicate logic system supports both simple rules as well as a complex rule having multiple conjunctions. Distributed representations have advantages of increased fault tolerance, graceful degradation of performance; neural plausibility, cognitive modeling and parallel distributed processing. The system besides showing above features allows the communication between these two connectionist systems and makes it possible to access the information of attributes and corresponding values from the connectionist semantic network for the entities used in the connectionist predicate logic system.

Categories and Subject Descriptors

I.5.1 [**Pattern Recognition**]: Models - *neural nets.*

General Terms

Design, Reliability, Experimentation.

Keywords

Coarse-coding, Connectionist, Reasoning, Fault Tolerance, Semantic Network.

1. INTRODUCTION

Traditionally reasoning systems using predicate logic have been implemented using symbolic methods of artificial intelligence. Connectionist methods of implementation of reasoning systems form an alternative paradigm and this is called sub-symbolic approach. Among the connectionist systems they use two types of representational schemes. They are 1) localist and 2) distributed representational schemes. Localist representational schemes represent each concept with an individual unit or neuron. In the distributed representational schemes [2] each unit or neuron is used in representation of multiple concepts and multiple units or neurons are used to represent a single concept. Subsymbolic systems for reasoning which are described in the literature used localist representations. The connectionist inference system *SHRUTI* [6], [7] described a localist method where temporal synchrony was used to create bindings between variables and entities they represent. A variable *x* of the predicate *give(x,y,z)* is getting bound to an entity *d* if the nodes representing them fire during the same phase of time during a predicate's activation period *T*. The time period *T* is divided into three phases *p1, p2* and *p3* during which synchronous firing of variable *x, y* and *z* and entity nodes they bound respectively takes place. *CONSYDERR* [1] described a localist method for variable binding and forward reasoning. Since, these systems used localist representations, advantages of distributed representations are not obtainable by them. In our previous works [3], [4] and [5] we have described the design and development of a connectionist predicate logic reasoning system and a connectionist non-monotonic reasoning system. In this present work, we describe the design and implementation of a connectionist knowledge based system which integrates a connectionist predicate logic reasoning system and a connectionist semantic network. The system uses distributed coarse-coded representations. The organization of the paper is as follows. In section 2, we describe the connectionist predicate logic reasoning system. In section 3, we describe the connectionist semantic network and in section 4, we describe the interconnecting connectionist network between these two systems. In section 5 we describe testing and verification and in section 6 conclusions.

2. CONNECTIONIST PREDICATE LOGIC REASONING SYSTEM

Connectionist predicate logic reasoning system represents and reasons with predicate logic rules and facts. Following are rules and facts we use.

1. *give(x, y, z)→ own(y, z);*
2. *own(y, z) → candonate(y, z);*
3. *own(y,z) ∧ wantstobuy(w, z) ∧ hasrequiredmoney(w, m)→ cansell(y,w,z);*
4. *give(John,Mary,Book-1);*
5. *give(John,Chris,Book-2);*
6. *wantstobuy(Walter,Book-2);*
7. *hasrequiredmoney(Walter,Money);*

In our rules and facts base we have used variables *x, y* to denote specific persons and have used variable *z* to denote a specific book. Our system uses the above rules and facts base and makes inferences shown below.

1. *own(Mary,Book-1);*
2. *candonate(Mary,Book-1);*
3. *own(Chris,Book-2);*
4. *cansell(Chris,Walter,Book-2);*

2.1 Forward Reasoning using Connectionist System

In this subsection, we see how to accomplish the forward reasoning for predicate calculus facts and rules using neural networks which operate on coarse coded distributed representations. Each fact of predicate p_i is represented by a vector $\mathbf{v_{ij}}$. The vector $\mathbf{v_{ij}}$ is a *k* dimensional vector which stores the coarse coded representation of predicate fact. The different instantiations of predicate p_i are each represented by separate vector $\mathbf{v_{ij}}$ where j varies from *1* to *m* where *m* is the number of vectors in predicate p_i table.

We describe here, briefly with an example how forward reasoning using localist representations [1], [6] is made using a connectionist system. Let us consider the *rule 1:give(x,y,z)—> own(y,z)* from the knowledge base. The localist pattern for the LHS of rule 1 can be written as *0001 001 001 001 1*. The first 4 bit value denotes the predicate *give*, the next 3 bit value denotes an object getting bound to variable *x, 'John'*, the next 3 bit value denotes an object getting bound to variable *y* , *'Mary'* and the next value denotes, '*Book-1'*. The last bit indicates the truth value of predicate *give*. This instantiation will activate rule *1* and make variables on the right hand side of the rule *'y'* and *'z'* be assigned the values *'001'* and '*001*' representing the objects *'Mary'* and ' *Book-1'* respectively.

Table 1. Shows a sample of localist tuples used by predicate *give*

S.No of Tuple	215
Predicate 'id' code	00001000000
Localist Value of x	0000100000
Localist Value of y	0000100000
Localist Value of z	0000010000
Truth Value of Predicate	00001

Because of the rule activation the localist pattern representation for RHS will be *0010 001 001 1* denoting *own(Mary,Book-1)*. This triggers the rules whose left hand sides match RHS of rule 1 and through this forward chaining, forward reasoning using localist representations is accomplished. In *Table 1* and *2* we show samples of localist vectors for some of the predicates in the rule base.

Table 2. Shows a sample of localist tuples used by predicate *Cansell*

S.No of Tuple	46
Predicate 'id' code	00000001000
Localist Value of y	0000000010
Localist Value of w	0000100000
Localist Value of z	0000010000
Truth Value of Predicate	00001

2.2 Obtaining Coarse-coded Distributed Representations from Localist Representations

Consider the following tuple from the localist representation table of predicate *give(x,y,z), '00001000000 0000000001 0000000010 0000000010 00001'.*

We view the above vector as being kept in overlapping coarse zones of length of 4 consecutive bits and encode the zone as *1* if there is at least one *1* bit in that zone or else as *0*. We then consider next coarse zone and encode it as *1* or *0* following above method. We do this process left to right starting from the left most bit. We do this encoding process for above localist tuple to get the following coarse-coded tuple
' 01111000000 0000001111 0000011110 0000011110 01111'.
Coarse-coding can be applied when the number of 1's in the original string is sufficiently sparse. If the number of 1's in the original string is not sufficiently sparse then coarse-coded string when decoded will not yield the original string. This is the reason we have chosen a 5 bit string to denote the truth value of predicate(in which first 4 bits were kept as zeros). The reason the coarse-coding could be applied successfully to our reasoning problem is that localist representations of instantiated predicates were sufficiently sparse with regard to distribution of 1's.

Table 3. Shows a sample of Coarse-code Representation of data tuples used by predicate *give*

S.No of Tuple	215

Predicate 'id' code	01111000000
Value of x	0111100000
Value of y	0111100000
Value of z	0011110000
Truth Value of Predicate	01111

Coarse-coding increases the information capacity [2] by increasing the number of units active at a time compared to localist codes which have sparsely populated 1's. The amount of information conveyed by a unit that has a probability *p* of being '*1*' is *– plog (p) – (1 - p)log(1 - p)*. We obtain the coarse-coded representations of tuples for all the predicates in the *rule base* using the above described method. We show here a sample of coarse-coded representations of the tuples for predicates *give* and *cansell* in the rule base.

Table 4. Shows a sample of Coarse-code Representation of Data Tuples used by predicate *cansell*

S.No of Tuple	46
Predicate 'id' code	00001111000
Value of x	0000011110
Value of y	0111100000
Value of z	0011110000
Truth Value of Predicate	01111

2.3 Organization of Neural Networks in the Connectionist Reasoning System

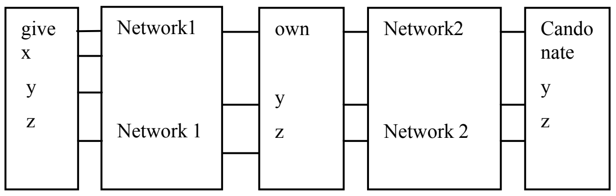

Figure 1. Neural Networks for processing rules 1 and 2

The neural networks shown accomplish the forward reasoning using the coarse-coded tuples. They generate inferences by firing rules from the rule base. Consider the neural networks shown in figure 1. When impressed on its inputs with one of the vectors $\boldsymbol{v_g}$ from the predicate table *give* the network 1 generates on its outputs a vector $\boldsymbol{v_o}$ from the predicate table *own*. This way the rule *give*(*x,y,z*) —> *own*(*y,z*) was processed. This in turn impresses on the inputs of network 2 to generate a vector $\boldsymbol{v_d}$ on its outputs. This processed the rule *own(y,z)* —> *candonate (y,z)*. These vectors are in coarse-coded form and denote a predicate fact. So we see the rules 1 and 2 are getting activated in a forward chaining fashion.

2.4 Variable Binding during processing of the Complex Rule

Consider the complex rule which involves multiple conjunctions.

own(y,z) ^ *wantstobuy(w,z)* ^ *hasrequiredmoney*(w,m) → *cansell(y,w,z)*.

This rule is processed by the connectionist architecture shown in figure 2. We use the vectors v_o, v_w and v_h from the predicate tables *own, wantstobuy* and *hasrequiredmoney* respectively. Though these are coarse-coded tuples their structure has the format of predicate code *p*, value of variable *1*, value of variable *2,*….. value of variable *n* and predicate truth value *T / F.* These constituents are distinguishable and hence they can be used directly. These constituents are in coarse coded form. We use these constituents to implement the complex rule under consideration. Component *z* is taken from both v_o and v_w and given to network *3*. This network generates truth value *T* or *F* depending on whether the values of variable *z* given to it are same or different. Similarly, component w is taken both from vectors, v_w and v_h and given to network *4*. This network generates truth value of *T* or *F* depending upon whether the values of variable *w* given to it are same or different. These truth values from network *3* and network *4* outputs are given to network *5* which outputs *T* if both of the truth values on its inputs are *true* else outputs *F*. The predicate code components of the vectors are given to another neural network *6* which outputs predicate code *p* for *cansell*. The values of *y, w* and *z* are passed on to the output lines as shown in figure 3 from the vectors v_o, v_w and v_w respectively. If network *5* output is 'T', the values of *y, w* and *z* are accepted as belonging to vector v_c of the predicate table of *cansell.* Using this method the variable binding problem has been solved, while processing the above complex rule, which is involving multiple conjunctions. Our task was to check whether the variable *w* belonging to both *wantstobuy* and *hasrequiredmoney* are binding to same value. Similarly, we had to check whether variable z belonging to *own* and *wantstobuy* are bound to the same value. We had accomplished these with networks *4* and *3* respectively.

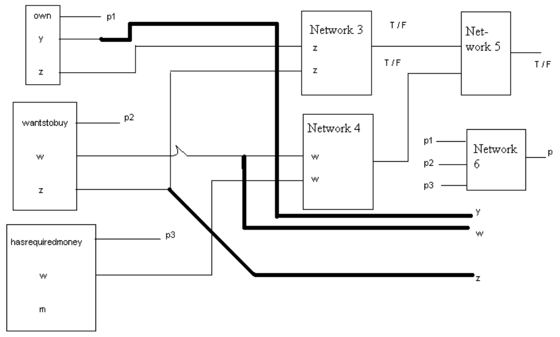

Figure 2. Neural Networks for processing rule 4

We have accomplished the variable binding task here using a divide and conquer strategy and distributed the total task to a set of neural networks which together accomplished the same. This approach can be similarly extended to handle even more complex rules which involve more number of conjunctions.

3. CONNECTIONIST SEMANTIC NETWORK

A semantic network consists of a number of nodes representing various entities. Each node in the semantic network represents an entity. Each entity has attributes and also corresponding values for those attributes. Here we use neural networks to implement semantic network. Figures 3 and 4 describe neural networks used to construct connectionist semantic network.

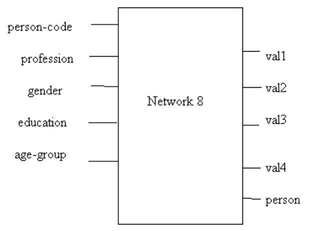

Figure 3. Depicts details of Inputs and Outputs of Neural Network 8

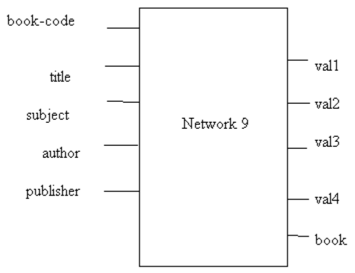

Figure 4. Depicts details of Inputs and outputs of Neural Network 9

We show in tables *5* and *6* a sample of the coarse-coded representations of the input and the output vectors for *neural network 8*. When given the coarse-coded input vector as shown in table *5* describing the identification code for a specific person and codes designating attributes *profession, gender, education and age-group,* neural network *8* produces on its outputs the coarse-coded vector shown in table *6,* with the codes representing the *values* for the above attributes of *profession, gender, education and age-group* respectively. This is a representation for information about a person identified by the *person-code* regarding the values of attributes *profession, gender, education and age-group.* It also denotes that entity represented by this neural network is an instance of *person*. Similarly, when given the coarse-coded input vector describing the identification code for a specific book and codes for attributes *title, subject, author* and *publisher* to neural network *9;* it produces on its outputs the coarse-coded vector with the codes representing the corresponding *values* for the above attributes of *title, subject, author* and *publisher* respectively. Neural network *9* is therefore representing information about a book identified by the *book-code* regarding its *title, subject, author* and *publisher*.

Table 5. Shows a sample of coarse-coded input data for Neural Network 8

person-code	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0
Coarse-coded representation for attribute ‘*profession*’	0 1 1 1 1
Coarse-coded representation for attribute ‘*gender*’	0 1 1 1 1
Coarse-coded representation for attribute ‘*education*’	0 1 1 1 1
Coarse-coded representation for attribute ‘*age-group*’	0 1 1 1 1

Table 6. Shows a sample of coarse-coded output data for Neural Network 8

Coarse-coded value for attribute *‘profession’*	0 0 0 0 1 1 1 1 0
Coarse-coded value for attribute *‘gender’*	1 1 1 1 0
Coarse-coded value for attribute *‘education’*	0 0 1 1 1 1 0
Coarse-coded value for attribute *‘age-group’*	0 1 1 1 1 0
Coarse-coded value designating entity *‘person’*	0 1 1 1 1

4. INTERCONNECTING NEURAL NETWORK

The neural network *7* is used to interconnect the connectionist predicate logic system and the connectionist semantic network.

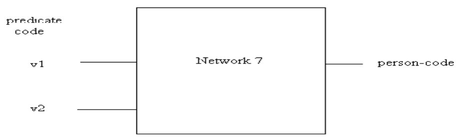

Figure 5. Inputs and outputs of Neural Network 7

Table 7. Shows a sample of coarse-coded input-output data for Neural Network *7*

Predicate ‘id’ code	0 1 1 1 1 0 0 0 0 0 0 0 0
Value of *v1*	0 0 0 0 0 0 0 0 0 1 1 1 1 0

Value of *v2*	0 0 0 0 0 0 0 0 0 0 0 0
Person-code output generated	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0

Neural network 7 takes on its input a coarse-coded pattern having a *predicate code* and the *value* of a variable denoting a person and it produces on its output a unique *person-code* identifying a person. In the figure 5, *v1* or *v2* can take as input the value of a variable denoting a person. Only one of these inputs, *v1* or *v2,* is activated at a time. If the value of a variable is input to *v1* then *v2* is given *all zeros* on its input and vice versa. This way *person-code* identifying a person is generated on the output of *Neural network 7.* This is explained below with an example.

For example, for the predicate *give(x, y, z)* consider the instantiation *give (John, Mary, Book-1).* The variables *x* and *y* denote persons *John* and *Mary* and the variable *z* denotes a book *book-1*. To generate unique person code corresponding to the person *John* we give following as inputs to the *neural network-7*: i) predicate code for *give* predicate to input *predicate 'id' code* ii) the coarse-coded value representing *John* to the input *v1* and iii) *all zeros* to input *v2*. Neural network-7 then generates as output a unique person code corresponding to *John*. Similarly, to generate unique person code corresponding to the person *Mary* we give following as inputs to the *neural network-7*: i) predicate code for *give* predicate to input *predicate 'id' code* ii) *all zeros* to input *v1* and iii) the coarse-coded value representing *Mary* to the input *v2. Neural network-7* then generates as output unique person code corresponding to *Mary*. In our rules and facts base, coarse-coded values given to the variables *x* and *y* denote specific persons. Same coarse-coded pattern, say, *0111100000* when assigned respectively to different variables *x* and *y* in predicate *give(x, y, z)* could be denoting different persons, say, *John* and *Mary.* This is because, entities denoted by the values for *x* and *y* variables are chosen independently. Since, *John* is bound to the first occurring variable *x* in predicate *give(x, y, z)* coarse-coded value representing *John* is given as input to *v1* in the neural network-7 to generate unique person code for *John.* Similarly, since, *Mary* is bound to the second occurring variable *y* in predicate *give(x, y, z)* coarse-coded value representing *Mary* is given as input to *v2* in the *neural network-7* to generate on neural network-7 output, unique person code for *Mary.* So for generating the unique person code corresponding to *John* or *Mary* we not only consider the coarse-coded value denoting the person, say *John* or *Mary* but also consider the position of the variable in the predicate (1^{st} or 2^{nd}) and accordingly input the coarse-coded value representing the person respectively to *v1* or *v2* input of *neural network-7*. This is done for values of *x* and *y* variables which denote persons. We show in table 7, sample of the coarse-coded representations of the input and the output vectors for *neural network 7*. This unique *person code* generated as output by *neural network 7* is used for giving as an input to the *person-code* input of *neural network 8*.

For the variable *z*, whose value denotes a book and since it is the only variable in the knowledge base whose value denotes a book the value of the variable itself serves as a unique *book-code* for giving as an input to the input *book-code* of *neural network 9*.

5. TESTING AND VERIFICATION

In table 8 we show the details of neural networks used. The neural networks in table 8 are feed forward neural networks using back-propagation algorithm.

The connectionist predicate logic reasoning system had performed reasoning satisfactorily for the reasoning task described in the knowledge base shown in section II. The connectionist semantic network which was implemented by networks 8 and 9 was verified to retrieve information about attributes and the corresponding values of *persons* and *books* respectively. The connectionist knowledge based system which is the result of integration of connectionist predicate logic system and connectionist semantic network, allowed communication between these two connectionist systems and made it possible to access the information of attributes and corresponding values from the connectionist semantic network for the entities used in the connectionist predicate logic system. For example, for the entity *john* which is used by connectionist predicate logic system with the predicate *give*, we can access from the connectionist semantic network the values for the attributes *profession, gender, education* and *age-group* using the integrated system. That means for every entity used by the connectionist predicate logic system we can store and retrieve attribute values relevant to the entity from the connectionist semantic network.

Table 8. Shows the details of neural networks used

Network	No. of input units	No. of hidden units	No. of output units
1	66	56	52
2	52	5	52
3	20	10	5
4	20	10	5
5	10	5	5
6	33	25	11
7	39	7	44
8	64	6	32
9	44	6	48

Secondly, to observe the fault tolerance property of the coarse-coded connectionist reasoning system we completed the following tests. The coarse-coded connectionist reasoning system was compared for error tolerance under noise conditions with a localist representation based connectionist reasoning system (which was having identical number of input, hidden and output units for its neural networks).

Table 9. Shows the details of test 1

Test 1	No. of training patterns	No. of test patterns	No. of output patterns correctly

			generated
Localist reasoning system	216	108	60
Coarse-coded reasoning system	216	108	89

In the test 1, neural network *1* with *66* input units, *56* hidden units and *52* output units was trained with *216* input-output patterns. *108* of these training patterns were made test patterns after introducing *1* bit error at a random location in each of the input patterns leaving the output patterns unchanged. If output pattern is correctly produced by the neural network *1* despite having a *1* bit error in the input side of the test pattern, it indicates the fault tolerance property of the connectionist system. Results of the test 1 are as shown in table 9.

In the test 2, neural network *1* was trained with *750* input-output patterns. 300 of these training patterns were made test patterns after introducing a *1* bit error at a random location in each of the input patterns leaving the output patterns unchanged. If output pattern is correctly produced by the neural network *1* despite having a *1* bit error in the input side of the test pattern, it indicates fault tolerance property of the connectionist system. Results of the test 2 are as shown in table 10. Tests were performed using the SNNS simulator. The coarse-coded reasoning system was found to be much more fault tolerant to errors compared to localist reasoning system as was indicated by the tests performed.

Table 10. Shows the details of test 2

Test 2	No. of training patterns	No. of test patterns	No. of output patterns correctly generated
Localist reasoning system	750	300	207
Coarse-coded reasoning system	750	300	274

6. CONCLUSIONS

We have designed and implemented a connectionist knowledge based system which integrates a connectionist predicate logic reasoning system and a connectionist semantic network and allows communication between them. The system makes it possible to access the information of attributes and corresponding values from the connectionist semantic network for the entities used in the connectionist predicate logic system making it an integrated system. Subsymbolic systems which use neural networks are brain inspired systems. Subsymbolic systems can show the properties of fault tolerance, graceful degradation in performance under error conditions and parallel distributed processing which are also the characteristics found in the human brain. Symbolic models while describing cognitive tasks like reasoning do not show above characteristics found in human brain. Hence, subsymbolic models are useful for modeling cognitive tasks like reasoning. Also, human brains are known for using distributed representations. In this work, we designed and developed a system with subsymbolic architecture using distributed coarse-coded representations to perform predicate logic reasoning and for realizing a semantic network. The knowledge based system used distributed coarse-coded representation vectors to represent instantiated predicates used by predicate logic system and for representing the attributes, corresponding values for the entities in the semantic network. Besides performing the functions of predicate logic reasoning and accessing information from the semantic network the system also showed the characteristics of fault tolerance, graceful degradation in performance under error conditions and parallel distributed processing.

7. REFERENCES

[1] Browne, A., and Sun, R. Connectionist inference models. Neural Networks. 14: 2001, 1331-1355.

[2] Hinton, G.E., McClelland, J.L., and Rumelhart, D.E. Distributed representations. Parallel Distributed Processing: Explorations in Microstructure of Cognition. 1986, 1:77-109, , MIT Press, Cambridge.

[3] Sanjeevi, S.G., and Bhattacharyya, P. Non-Monotonic reasoning with connectionist networks using coarse-coded Representations. In Proceedings of the 5th IEEE International conference on Machine Learning and Cybernetics (Dalian, China, August 13th-16th, 2006). Vol. 5, pp. 3048-3052.

[4] Sanjeevi, S.G., and Bhattacharyya, P., A Connectionist Model for Predicate Logic Reasoning using Coarse-coded Distributed Representations. In Proceedings of the 9th International conference on Knowledge-based & Intelligent Information and Engineering Systems(KES 2005), Melbourne, Australia, Lecture Notes in Artificial Intelligence, Springer-Verlag, Berlin Heidelberg, No. 3682*,* 732-738.

[5] Sanjeevi, S.G., and Bhattacharyya, P., A fault tolerant connectionist model for predicate logic reasoning, variable binding: using coarse-coded distributed representations. WSEAS Transactions on Systems, 4, 4, (Apr. 2005), 331-336.

[6] Shastri, L. Advances in SHRUTI: a neurally motivated model of relational knowledge representation and rapid inferencing using temporal synchrony. Applied Intelligence. 11(1), 1999, 79-108.

[7] Wendelken, C., and Shastri, L.. Multiple instantiation and rule mediation in SHRUTI. Connection Science. 16, 2004, 211-217.