Dynamics of Emergent Computation in DUAL

Boicho Kokinov[1 2] **Vassil Nikolov**[1 2] **Alexander Petrov**[2]

[1] *Institute of Mathematics*
Bulgarian Academy of Sciences
Bl. 8, Acad. G. Bonchev Str.
Sofia 1113, Bulgaria

[2] *Department of Cognitive Science*
New Bulgarian University
21, Montevideo Str.
Sofia 1635, Bulgaria

e-mail: kokinov@cogs.nbu.acad.bg

Abstract. Human cognitive processes exhibit three important qualities: flexibility, efficiency, and context sensitivity. The paper argues that these properties can be achieved in cognitive models by a combination of emergent computation and computational dynamics. We discuss the possibilities of building such models on the basis of DUAL — a distributed cognitive architecture. Computation in DUAL emerges from the collective behaviour of a great number of specialised agents running in parallel and interacting through a network of relations. The agents work at individual speeds reflecting their relevance to the context. Moreover, new agents and new relations can be created in the course of computation. In this way the particular set of agents engaged in a cognitive task and the pattern of their interaction is formed dynamically. An example of such *dynamic emergent computation* — mapping between two symbolic structures — is discussed in more detail.

1. Introduction

The computations performed by a cognitive system should be *efficient* enough as they are performed in real time. They must be *flexible* enough to be able to cope with the variety of situations the system faces. In open real-world situations it becomes impossible to meet this requirement if one has to foresee all cases that may occur and provide for their handling *in advance*.Therefore, it is very important that the system possesses the capability of *dynamically adapting* its process of computation to any situation it faces, whether predicted beforehand or not. Moreover, the system should be able to choose different ways of accomplishing the same task under different circumstances, picking up one that is more appropriate and/or more efficient. We favour such a style of computation, and we use for it the name *context-sensitive symbolic processing*.

In A. Ramsay (Ed.) (1996), Artificial Intelligence: Methodology, Systems, Applications (pp. 303–311). Amsterdam: IOS Press. http://alexpetrov.com/pub/ambr96/

This type of processing is in contrast with the approach in conventional computing which we would call *rigid processing. Rigid computation* is primarily characterised by the existence of an *a priori specification* of its task, and has the following advantages:

• it is possible to achieve guaranteed maximum (or at least high) *efficiency* of the processing unit(s) doing the symbolic computation;

• rigid processing is *stable* in the sense that it can be relied upon to always produce the desired effect, in accordance with the corresponding specification (its *a priori* existence means that it can be used for the decision whether to start a particular computation);

• it is possible to ensure *computational consistency* with rigid processing, i.e. that computational resources are spent in a systematic way for the achievement of a single goal or a coherent set of goals.

Rigid computation alone cannot meet the requirements described in the first paragraph of this section, and a cognitive system will do so only if it can exhibit *emergent processing* [4]. The principal feature of a symbolic process which can be characterised as *emergent computation* is that it has no explicit *a priori* specification of either what it computes or how it computes it. In this case, it may be possible to interpret the collective behaviour of all processing units as a particular computation only after the process is finished.

As a consequence of the above, emergent computations are characterised by the following:

• processing is *distributed* over a set of parallel and autonomous processing units without a 'central authority' (otherwise, if there is only a single unit, or one unit controls the others, the description of the processing it does would be an explicit *a priori* specification of the whole system);

• all explicit processing is done *locally* and each unit can interact directly only with a small limited set of other units (in other words, no single unit may be capable of communicating with all processing units engaged in a particular task because otherwise this would enable it to establish central control over their operation);

• the mechanisms of emergent processing should promote *consistency* among the various processing units (to avoid chaotic and uninterpretable collective behaviour).

It should be noted that the characteristic features of emergent computation can vary across a continuous spectrum. Each may be manifested to a different extent in a particular model. Explicit *a priori* specifications may exist in various degrees, so that there are no clear-cut boundaries between 'rigid' and 'emergent' computations.

What is computed in the case of emergent processing is determined by:

• the organisation of the computational system, comprising *(i)* the specifications of local rigid computations on a per-unit basis (there may be homo- and/or heterogeneous collection of units), and *(ii)* the specification of unit-to-unit connectivity;

• dynamic changes in that organisation, defined as the changes in the set of units actually involved in a particular computation and the changes in the actual patterns of interactions between these units.

These considerations lead to two additional very important aspects that determine to what extent a process is emergent:

• *learning and self-organisation*, which are concerned with slow *long-term changes* in the organisation of the system whereby accumulation of experience leads to improvement of performance;

• *computational dynamics*, concerned with quick *short-term changes* in the organisation of the computation that reflect the particular task and the particular context. Computational dynamics is especially important for achieving a reasonable tradeoff between flexibility on one hand, and efficiency and context-sensitivity on the other.

Reviewing some architectures that exhibit typical behaviour of emergent computation, we can classify them according to two criteria following from the previous paragraph: *(i)* the degree to which they can learn and self-organise, and *(ii)* the degree to which they exhibit computational dynamics (Fig. 1).

Cellular automata [2], for example, have a fixed organisation, which does not change either in the course of computation or over longer periods.

Neural networks [18] do not change in the course of computations either, but are capable of learning a pattern of interaction between the processing units. They cannot, however, be taught either the computations the units do or the number of units and the network topology – all of these are designed entirely by the creator of the network.

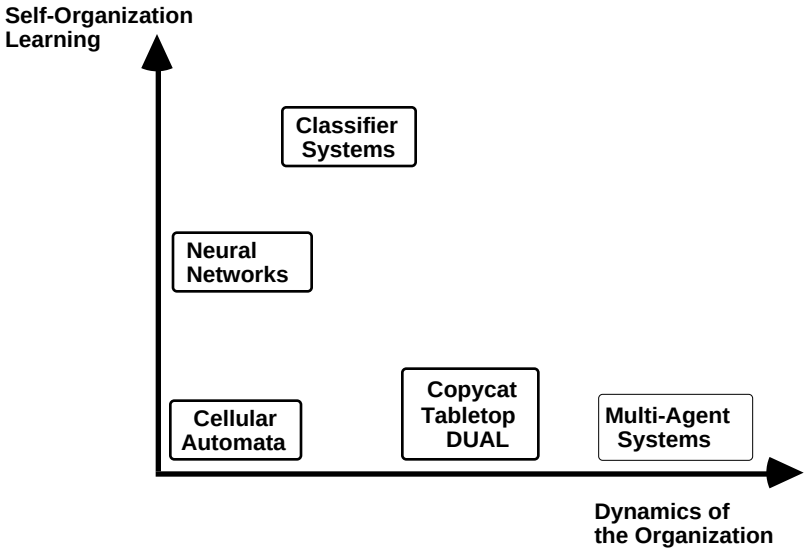

Figure 1. Long-term and short-term dynamics of emergent computation models.

Classifier systems [1, 10] have extensive mechanisms for learning and self-organisation that improve both individual computational capabilities (by means of genetic algorithms) and connectivity (by means of the bucket brigade algorithm). All these changes, however, are slow and affect the long-term organisation of the system; there are no mechanisms for dynamic change in the course of computations (with the exception of a recently developed variety, PI [11], that has a mechanism for dynamic activation of rules).

Unlike the latter, *classical multi-agent systems* [3, 7] for modelling social interaction do not possess any learning and self-organising capabilities but are equipped with very flexible mechanisms for establishing dynamic interactions between agents, including negotiation, cooperation, grouping, competition, etc. Such a system, however, is too complex to be considered a single computational system: the idea behind these systems is that they consist of many intelligent individuals, while we are interested in an architecture for individual human cognition.

The architecture underlying the *COPYCAT* and *TABLETOP* [5, 8, 9, 17] models of analogy-making, though lacking in learning capabilities, has a number of dynamic characteristics: the concepts in the Slipnet have varying activation levels, the structures in the Workspace have varying strengths, the processing units ('codelets') have varying 'urgencies.' However, the topology of the Slipnet is fixed and does not change during the functioning of the system.

The cognitive architecture *DUAL* [14, 15], the focus of this paper, has (for the time being) very limited learning capabilities, but is provided with mechanisms that determine dynamically which units take part in each computation and is capable of changing dynamically the pattern of interaction between the processing units by building temporary links and nodes. These characteristics make it flexible and context-sensitive and at the same time rather efficient.

2. Architectural Principles of Dual

From the perspective of this paper, DUAL was designed primarily with these three qualities in mind: *flexibility*, *context sensitivity* and *efficiency*, which are desirable for any computational model of cognition.

To achieve these qualities, DUAL's design has had as its main architectural objectives the availability of *emergent computation* and *computational dynamics*. For the combination of these two, we use the name *dynamic emergent computation.* The following paragraphs list the basic architectural principles according to which DUAL is designed.

Distributed computation. All information processing in DUAL is done as computation distributed over (a large number of) autonomous agents (individual entities capable of storing and processing information) running in parallel. Parallelism arises naturally with many cognitive processes including perception (simultaneous receiving of input from different modalities), action (simultaneous motor control of different parts of the body, simultaneous

performance of different actions), problem solving (simultaneous perceiving of the problem and the environment, retrieving information from memory, and 'proper' reasoning). Parallelism allows to achieve complex behaviour by the combination of relatively unsophisticated components (agents), and is a major factor in increasing efficiency.

Local rigid computations by heterogeneous agents. Each rigid computation in DUAL is local and performed by an individual agent; each agent has an agent-specific set of such computations attached to it. DUAL agents are typically much more complex than connectionist processing units (artificial neurons) or rules in classifier system, being able to do rather elaborate symbolic processing. On the other hand, there are no 'general-purpose' agents, and the architecture comprises a heterogeneous multitude of specialised agents, each being significantly less complex than a universal Turing machine or an agent in classical multi-agent system. It is only the collective behaviour of all agents that produces a cognitive system.

Emergent global computation. All global symbolic processing is done as emergent computation, arising from the parallel work and the (cooperative or competitive) interaction of the local rigid computations done by the individual agents. There is no central authority exercising global control over the system .

Dynamically varying availability of agents. In different contexts, different agents may be 'available' to the system and they may contribute to overall processing in different degrees. Each agent operates at a specific speed which varies across agents and over time. The fact that the speed may vary means that the interval between two consecutive interactions of that agent depends, from its point of view, not only on the nature of the computation it performs but also on another factor – its availability. Similarly, information stored in an agent may or may not be accessible to other agents in different degrees. The idea is that the availability of an agent depends on its relevance to the context, with very relevant agents producing results quickly and irrelevant agents running slowly or not at all. In this way the collection of agents actually contributing to the computation and the timing of their contributions are determined dynamically by the context.

Dynamic network organisation of agents. Agents are organised in a network representing all possible patterns of interactions between them. The organisation of this network may be part innate and part produced by experience. Moreover, dynamic topological changes can take place in the network so that it reflects the particular cognitive task, e.g. the operation of some agents leads to the creation of new links or new nodes in the network. In this way the patterns of interactions between the agents become dynamic and context-sensitive.

3. Agents and Mechanisms in Dual

This section describes how DUAL is constructed in accordance with the principles described in the previous section.

3.1. The nature of agents

The principle of local rigid computation is realised in DUAL by having a large number of agents, each of which operates from a computational point of view as a specialised symbolic processor with local memory which can perform several agent-specific symbolic-processing tasks. From the point of view of the system's knowledge, it is distributed over the agents, each representing a specific piece of it (e.g., a concept, an event, an object, a rule, etc.). Representationally, each agent carries both declarative and procedural knowledge (the latter being executable by the same agent), and is technically a frame-like structure. The slots of the frame are part of the agent, but their fillers are represented by separate agents.

As an illustration of the above, consider the agent representing the relation 'in.' The declarative knowledge in it includes the fact that this '*is-a'* binary spatial relation. The agent-specific procedural knowledge here supplies the method which can be used to check whether one object is 'in' another. The general procedural knowledge enables the agent to take part in marker-passing, i.e. receive a marker, store it, compare it to the ones received earlier, and pass it further along.

The availability principle is realised by equipping DUAL with a connectionist mechanism of spreading activation. Each agent has an activation level which determines dynamically its availability, i.e. the speed of its symbolic processor and the accessibility of its declarative knowledge: the higher the level of activation, the greater the availability. From the point of

view of this mechanism, each agent is a connectionist processor, continually computing the activation level of the agent and spreading activation to other agents.

In more detail, the timing of each symbolic operation done by an agent, and hence the speed of its symbolic processor, depends on its activation in the following way. Let *a(t)* be the activation level at time *t, 0<a(t)<1*. Let $A(t) = \int_{t_0}^{t} a(\tau)d\tau$ (the operation starts at time t_0, $t>t_0$). *A(t)* is an increasing function of *t* and can be seen as the amount of activation at time *t* 'spent' at the node since the operation started. Each symbolic operation has a parameter $\alpha > 0$ associated with it which can be construed as 'activation fuel', the amount of activation that must be 'spent' by the agent to complete the operation; mathematically this can be expressed by stating that the duration of the operation is $t_1 - t_0 = A^{-1}(\alpha) - t_0$.

In all, DUAL agents are hybrid entities, each having an *s-component* and a *c-component*. The *s-component* takes care of the symbolic processing done by the agent, i.e. it contains its symbolic processor, memory, and (rigid) procedures. The *c-component* provides the functioning of the connectionist mechanism.

3.2. Network organisation of agents

The agents form a network that represents the typical patterns of interaction among them. Most of the links are permanent, but additional temporary links can be created dynamically in the computation process. The same network is used by the *s-component*s and by the *c-component*s.

From the perspective of *s-component*s, links can be used to send symbols and symbolic structures from agent to agent. The links have labels (recognisable by the symbolic processors) and represent semantic relations (e.g. *is-a, instance-of*) between the concepts and objects represented by the agents.

From a connectionist perspective, links are employed by the *c-component*s to spread activation. Besides the 'semantic' links, this also takes place along the so-called *a-links*, used to represent arbitrary associations and 'invisible' to the *s-component*s. For the *c-component*s, the labels of the links are not taken into account but the links are weighted ([-1, 1]).

The network formed by all permanent agents represents the long-term memory (LTM) . The working memory (WM) comprises all active agents (more precisely, those with above-threshold activations) and is formed by the active subset of LTM and by the active temporary agents. An agent is not available unless and until it becomes active, therefore agent interactions are restricted to the WM only.

3.3. Representing context and relevance to it

It must be noted that *context* does not exist as an explicit entity within DUAL; rather, it is given an implicit distributed representation over all concepts 'known' to the system (i.e. over LTM). The primary responsibility for maintaining DUAL's idea of the current context lies with the connectionist machinery which represents the context as the distribution of activations over the network of agents. If an agent is more relevant to the context, it will be more active, therefore it will be more available, and it will contribute to a greater extent to the overall performance of the system. If an agent is contextually relevant, others related to it (directly or indirectly) are also relevant, and, on the other hand, they will indeed receive activation from it along the links representing the respective relations. Ideally, therefore, the WM should comprise precisely those agents representing contextually relevant concepts. Besides, the level of activation provides a direct quantitative measure of the contextual relevance of each agent.

The way in which context information 'enters' DUAL is rather simple. DUAL 'is told' about the outside world by permanently activating nodes in LTM representing entities being 'perceived' by DUAL. These nodes play the role of *input nodes* at the particular moment. To have DUAL accomplish a certain task, e.g. solve a problem, *goal nodes* are also kept active. The rest of DUAL's idea of the context comes from the task it pursued prior to being given the current one. DUAL is not 'reset' between successive tasks, and due to a relatively low

activation decay rate there is an inertia of the activation patterns, so that each task influences the way DUAL operates even after it has been accomplished.

4. Mapping between Structures: An Example of Emergent Computation in Dual

Although DUAL has been applied to modelling of various cognitive processes, because of space limits only one example of emergent computation will be considered in this paper – the process of establishing a mapping between structures. This process is employed by various cognitive processes like perception, language processing, reasoning, etc.

4.1. The task of mapping.

The mapping process accomplishes the task of establishing a consistent system of correspondences between the elements of a pair of symbolic structures. The result can then be used to evaluate how similar the structures are and to transfer elements from one of them to the other.

The mapping produced should satisfy, to the greatest extent possible, the following constraints [12]:

- *structural constraint* – the pressure to find and use a mapping which constitutes (or is as close as possible to) a structural isomorphism [6];
- *semantic constraint* – the pressure to find and use correspondences primarily between semantically similar elements;
- *pragmatic constraint* – the pressure to find and use correspondences primarily for elements of higher importance (higher relevance to the task at hand).

4.2. Producing the mapping using a constraint satisfaction network.

In general, a great number of pairwise correspondences between elements of the mapped structures may be created, and these may be grouped into many possible consistent mappings. Obviously, the task arises of selecting the 'best' one among them, comprising the 'best' consistent set of hypotheses. This task is accomplished in DUAL by the relaxation of a constraint satisfaction network (CSN). This network is built of temporary nodes, each representing a single correspondence hypothesis, and temporary links between the nodes; the positive (excitatory) links among these represent mutual support between consistent hypotheses, and the negative (inhibitory) ones represent competition.

Figure 2 illustrates the mapping process by displaying the structures involved in it in a fragment of DUAL's memory.

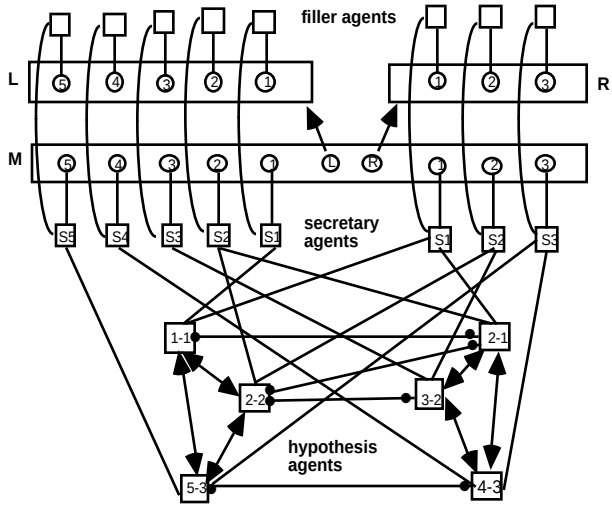

Figure 2.

In this simplified example, two frames: L (with 5 slots) and R (with 3 slots) are being mapped. The *map* frame, M, serves as the basis for building the CSN. For the slots in L and

R, M has corresponding slots labelled in the same way in its 'L' and 'R' parts, respectively. The fillers of these slots are specialised 'helper' agents called *secretaries* (S_i above), in the operation of which the construction of the CSN is localised. Each hypothesis agent in the CSN is labelled, in accordance with the slot-to-slot correspondence it represents, with two numbers, the first referring to a slot in L and the second to a slot in R. Excitatory links end with arrows and inhibitory ones with circles, the arrows of some excitatory links are omitted for simplicity.

The relaxation of the CSN is performed by DUAL's connectionist mechanism; it follows from this that relaxation takes place even while the CSN is being constructed, so that there is a temporary winner at any time. The support for each hypothesis is measured by the activation level of the corresponding node in the CSN. The winners among the hypotheses are selected by the secretaries and they construct the final result of the mapping process – the map.

4.3. The construction of the CSN as an emergent process.

Holyoak and Thagard [12] use a similar CSN in their solution of the mapping problem. There are, however, some differences between their approach and ours:

• they employ a rigid symbolic process to construct the CSN (scanning the predicate calculus descriptions of the pair of situations and producing the hypotheses sequentially in a straightforward order);

• hypotheses are produced for literally *all* possible correspondences (including semantically impossible and irrelevant ones), in a way that is computationally very demanding (so the model would not scale up well and appears cognitively implausible).

In contrast, the CSN in DUAL is constructed by an emergent context-sensitive process (described in the following paragraphs) combining the flexibility of the 'all correspondences' approach with a restriction of the hypotheses actually constructed only to ones relevant to the context, the number of these being quite limited. This approach is computationally less expensive as well as more plausible from a cognitive point of view.

Each time a hypothesis node is created, the secretaries 'serving' the slots that are put in correspondence are linked to that node. It is the job of the two secretaries involved to check if the newly created hypothesis is either *(i)* identical with an existing one, to avoid duplication, or *(ii)* in contradiction with an existing one, in which case an inhibitory link is created between the two (such is the case with [2-1] and [2-2] in Figure 2).

The construction of the CSN emerges from the parallel operation of many agents, each adding a new node and/or new links. Some of these agents create hypotheses in cooperation, enabling other agents to create other hypotheses supporting or supported by the ones created earlier, while other agents compete in creating mutually exclusive hypotheses or in extending the network in opposite directions.

4.4. Using semantic similarity to create hypotheses.

In contrast with other models (ACME, COPYCAT, TABLETOP), semantic similarity is computed dynamically by an emergent process of marker passing. This mechanism provides a context-sensitive measure of similarity. Essentially, two concepts (or instances thereof) will be similar if both can be considered particular cases of the same more general concept. If we take into account the fact that from different points of view a concept may have different 'superconcepts', the same criterion may be formulated in a more precise way as follows: two concepts will be similar if a point of view can be found from which both concepts are seen as having a common (not necessarily direct) superclass. Moreover, the relevance of that superclass to the current context (as measured by the activation level of its node) provides a quantitative measure of the similarity between the pair of concepts under consideration.

Semantic similarity in DUAL is computed by an emergent marker-passing process which proceeds along the following lines. It starts with marking the elements of the pair of structures (with different sets of markers for the two structures). Following that, each marked node locally processes any markers passed to it by passing them further along *is-a* and *instance-of* links over to its neighbours. Moreover, the speed of that processing depends on the activation level of that node; in particular, inactive nodes cannot pass their markers.

Whenever a node acquires two (or more) different markers, it establishes a link to the map agent M and communicates the situation of 'marker rendez-vous' to it. Then the map agent creates a semantic similarity hypothesis agent as a temporary node with temporary links to the secretaries of the original concept nodes.

4.5. Using structural similarity to create hypotheses.

Such hypotheses are created on the assumption that if two propositions are in correspondence, their predicates and arguments should also be in (pairwise) correspondences; the structural similarity hypotheses are precisely about the latter correspondences. These are created by structural correspondence (SC) agents; after a semantic similarity hypothesis is created, a SC agent is activated by linking it to the hypothesis node. The SC agent, in its turn, creates structural correspondence hypotheses for each predicate/argument; positive (supporting) temporary links are created between these new hypotheses.

4.6. Pragmatic support for the hypotheses.

The temporary nodes in the CSN are linked to the permanent nodes in LTM representing concepts, objects, etc. by temporary excitatory links (passing through the secretaries). Therefore each CSN node receives support (in the form of activation) from active permanent nodes linked to it. In this way pragmatically important (highly activated) concepts support correspondence hypotheses which involve these concepts and make them more relevant (active). Therefore, the relaxation of the CSN is pragmatically controlled and context-sensitive.

4.7. Dynamics and context-sensitivity of mapping.

In sum, the semantic similarity agents compete with the structure correspondence agents in the process of creating more and more nodes in the CSN. Precisely what nodes, in what order, and how many of them are created is determined by the processing speed of the different competing agents which in turn depends on their levels of activation. In this way the construction of the CSN is dynamic and context-sensitive.

The relaxation of the CSN is also sensitive to the general distribution of activation over the whole LTM and therefore is dynamic and context-sensitive as described in the previous subsection. Generally, this means that different mappings will be established between the same pair of structures in different contexts.

5. Implementations and Applications of the Dual Architecture

There are two different implementations of DUAL. The old one is developed for IBM PC computers and relies on non-standard extensions to Lisp supported by the Golden Common Lisp environment. In this version, the parallel operation of agents is simulated by conventional multitasking, associating a stack group with each running agent. The speed at which the agents run is varied by controlling the number of Lisp program steps that each agent is allowed to execute before a switch to the stack group of another agent takes place, which leads to a highly implementation-dependent measure of simulated time.

This version of DUAL has been used in modelling similarity judgement [16], analogy-making [13], and deductive reasoning [15].

A new portable Common Lisp version is currently under development. In this version, an implementation-independent measure of simulated time is provided. Simulation of parallelism is achieved by having the agents perform their symbolic operations by executing them as if instantaneously, at precisely that moment in simulated time when the operation should be completed according to the computed duration of the operation.

6. Conclusions

DUAL is an attempt to build an architecture such that cognitive models based on it are at the same time flexible, context-sensitive, and efficient. Emergent computation is an important

instrument for the achieving of these qualities. We believe, however, that it is not sufficient by itself, and should be accompanied by computational dynamics to obtain good results. Emergent computation is the subject of a vast body of research, but only a relatively small part of it can be characterised as being devoted to dynamic emergent computation. (Besides DUAL, COPYCAT/TABLETOP are the only such cases known to us). We expect that the approach of dynamic emergent computation will be sustained or refuted after extensive experimentation with different architectures designed along these lines.

Acknowledgments

This research has been partially supported by Bulgarian National Science Fund grant OHN406.

References

[1] Booker, L., Goldberg, D. & Holland, J. (1989). Classifier Systems and Genetic Algorithms. Artificial Intelligence, 40, 235-282.
[2] Burks, A.W. (ed.) (1970). Essays on Cellular Automata, University of Illinois Press, Urbana, IL.
[3] Castelfranchi, C & Werner, E. (1994). Artificial Social Systems. Springer-Verlag, Berlin.
[4] Forrest, S. (ed.) (1990). Emergent Computation. Physica D, vol 42.
[5] French, R. (1995). Tabletop: An Emergent, Stochastic Computer Model and a Theory of Analogy-Making. MIT Press, Cambridge, MA.
[6] Gentner, D. (1983). Structure-Mapping: A Theoretical Framework for Analogy, Cognitive Science 7(2), pp. 155-170.
[7] Gilbert, N. & Doran, J. (1994). Simulating Societies.UCLPress, London.
[8] Hofstadter, D. (1995). Fluid Concepts and Creative Analogies. Basic Books, NY.
[9] Hofstadter, D. & Mitchell, M. (1994). The Copycat Project:. In: Holyoak, K. & Barnden, J. (eds.) *Analogical Connections, Adv.in Connectionist and Neural Computation Theory,* vol. 2, Norwood, NJ: Ablex .
[10] Holland, J. (1975). Adaptation in Natural and Artificial Systems. Ann Arbor: University of Michigan Press.
[11] Holland, J., Holyoak, K., Nisbett, R., Thagard, P. R. (1986). Induction, Cambridge, MA: MIT Press.
[12] Holyoak K. & Thagard P. (1989). Analogical Mapping by Constraint Satisfaction. Cognitive Science vol. 13, pp. 295-355.
[13] Kokinov, B. (1994) A Hybrid Model of Reasoning by Analogy. Chapter 5. in: K. Holyoak & J. Barnden (eds.) Analogical Connections, Adv. in Connectionist and Neural Computation Theory, vol.2, Ablex .
[14] Kokinov, B. (1994) The DUAL Cognitive Architecture: A Hybrid Multi-Agent Approach. In: A. Cohn (ed.) Proceedings of ECAI'94. John Wiley & Sons, Ltd., London.
[15] Kokinov, B. (1994) The Context-Sensitive Cognitive Architecture DUAL. In: Proceedings of the 16th Annual Conference of the Cognitive Science Society. Erlbaum, Hillsdale, NJ.
[16] Kokinov, B. (1992) Similarity in Analogical Reasoning. In: V.Sgurev & B. du Boulay (eds.) Artificial Intelligence III: Methodology, Systems, Applications, Elsevier Science Publ.
[17] Mitchell, M. (1994). Analogy as Perception. MIT Press, Cambr., MA.
[18] Rumelhart D., McClelland J. and PDP R.G. (1986). Parallel Distributed Processing, vol.1: Foundations, Cambridge, MA: MIT Press.