ELSEVIER

BioSystems 64 (2002) 127–140

Bio Systems

www.elsevier.com/locate/biosystems

The brain-machine disanalogy revisited

Bernard P. Zeigler *

ECE Department, Arizona Center for Integrative Modeling and Simulation, The University of Arizona, Tucson, AZ 85721, USA

Received 31 March 2001; received in revised form 12 June 2001; accepted 27 June 2001

Abstract

Michael Conrad was a pioneer in investigating biological information processing. He believed that there are fundamental lessons to be learned from the structure and behavior of biological brains that we are far from understanding or have implemented in our computers. Accumulation of advances in several fields have confirmed his views in broad outline but not necessarily in some of the strong forms he had tried to establish. For example, his assertion that programmable computers are intrinsically incapable of the brain's efficient and adaptive behavior has not received much examination. Yet, this is clearly a direction that could afford much insight into fundamental differences between brain and machine. In this paper, we pay tribute to Michael, by examining his pioneering thoughts on the brain-machine disanalogy in some depth and from the hindsight of a decade later. We argue that as long as we stay within the frame of reference of classical computation, it is not possible to confirm that programmability places a fundamental limitation on computing power, although the resources required to implement a programmable interface leave fewer resources for actual problem-solving work. However, if we abandon the classical computational frame and adopt one in which the user interacts with the system (artificial or natural) in real time, it becomes easier to examine the key attributes that Michael believed place biological brains on a higher plane of capability than artificial ones. While we then see some of these positive distinctions confirmed (e.g. the limitations of symbol manipulation systems in addressing real-world perception problems), we also see attributes in which the implementation in bioware constrains the behavior of real brains. We conclude by discussing how new insights are emerging, that look at the time-bound problem-solving constraints under which organisms have had to survive and how their so-called 'fast and frugal' faculties are tuned to the environments that coevolved with them. These directions open new paths for a multifaceted understanding of what biological brains do and what we can learn from them. We close by suggesting how the discrete event modeling and simulation paradigm offers a suitable medium for exploring these paths.

Keywords: Brain-machine disanalogy; Programmability; Computational complexity; Discrete event abstraction; Temporal cognitive models; Fast and frugal heuristics

1. Introduction

Michael Conrad believed strongly that there is much more to information processing than is manifested in man-made artefacts, when com-

* www.acims.arizona.edu.
E-mail address: zeigler@ece.arizona.edu (B.P. Zeigler).

0303-2647/02/$ - see front matter
PII: S0303-2647(01)00181-2

Table 1
Aspects of (dis)analogy from theory of computation

Aspect of analogy	Digital computer	Conrad's conception of brain
Activity	Programmable from outside (passive)	Self-organizing (active tendency to learn and evolve)
Exploitation of interactions	Highly constrained (n interactions at most)	Highly interactive (tends toward $n^2/2$ interactions)
Structural programmability	Structurally programmable, low evolutionary adaptability, low computation efficiency	Structurally non-programmable, high evolutionary adaptability, high evolutionary efficiency

pared with that carried out by biological organisms. He argued that some of this difference is not only evident with regard to the computer technology of the moment, but is fundamental and will not be affected by advances in this technology. More than a decade after he put the brain-machine disanalogy to paper, it is fitting to look at Michael's insights from the perspective of the tremendous advances in certain aspects of computer technology and also the more modest advances in understanding of biological information processing. In honor of Michael, I will revisit his thoughts from my own limited perspective, admitting at the outset, that I will not attempt to include Michael's recent successes in exploiting molecular information processing. I will try to cast some light on Michael's thoughts that might be derived from recent results in such fields as computational theory, cognitive psychology, neuroscience, and modeling and simulation.

Underlying the brain-machine disanalogy paper are such questions as:

- Are computers and organisms different as computational devices based on fundamental theory?
- How do computers and organisms differ in their exploitation of the natural processes available for information processes?
- How can we bring computers closer to organisms in capability?
- What are the appropriate abstractions and models to help reduce the gap?

Michael's answers were summarized in Table 1 of his paper (Conrad, 1989).

2. Role of the theory of computation

The part of the Michael's Table 1 relating to theory of computation is reproduced here as Table 1. Michael's conception of computation was derived from the Turing–Church Thesis that all computer formalisms capable of universal computation are equivalent to the Turing machine formulation. His arguments relating computational complexity to fundamental particle interactions also fall within this paradigm.

As the table shows, Michael believed that the brain is an active, self-organizing biologically evolved system that is highly adaptable and efficient in its computation. In contrast, the digital computer is intrinsically a passive mechanism that must be programmed from the outside to perform useful functions. Inevitably, some of the underlying computational resources must be devoted to enable such programmability. Therefore, a computer cannot exploit the full potential of its resources and so must be inherently less efficient and less adaptable than the brain. In fact, to emphasize this difference as an intrinsic disanalogy between computer and brain, Michael formulated the *tradeoff principle* in Conrad (1988), which states that no system can be at once highly structurally programmable, highly evolutionary efficient, and highly computationally efficient. Roughly, his argument says that for a physical system of n particles, the available computational resources are the n^2 force interactions among the particles. Structural programmability, meaning the existence of a 'user's manual', constrains these interactions to a subset of order n, and these are

not enough to handle adaptation and computational efficiency which typically grow with order n^2. The brain however, devotes almost all of its n^2 interactions to computational efficiency and adaptability, leaving few to enable it to be programmed from the outside.

Such a tradeoff, and particularly its underlying concept of structural programmability, is an original and unique contribution of Michael's that has not attracted the attention it deserves. I will consider it now. To provide some of the basic definitions, the following are almost verbatim extractions from Conrad (1988, 1989).

A real system is *programmable* if we can prescriptively communicate a program to it. Such a system is *effectively programmable* if it is possible to communicate the program to it in an exact manner (without approximation) using a finite set of primitive operations and symbols. Effective programmability requires operation in a sequential mode, otherwise unanticipated conflicts would always be possible. Further, an effectively programmable system is *structurally programmable* if it is effectively programmable and its program is mapped by its structure. An example of structural programming is the McCulloch–Pitts neural network formalism, where a finite set of component types and a finite set of connection rules allow one to specify a well-formed network. Here, the program is given by the structure of the network (the components and their connections). Compilers, translators, and interpreters are other examples of support for structural programmability where they operate on the structures of symbols and their juxtaposition is common in contemporary programming languages. However, most physical systems are not structurally programmable. A falling apple computes Newton's laws but is not effectively programmable. It is logically possible that the brain could be simulatable by a digital computer, but nevertheless, not be decomposable into building blocks that correspond to elements of a language. In other words, the brain need not be structurally programmable even if a universal Turing machine could replicate its behavior.

In this way, Michael sets up structural programmability as the critical differentiator between brain and machine. On one side, we have structurally programmable devices, of which current digital computers are examples par excellence. And on the other, we have brains, which are not structurally programmable and hence, supposedly not subject to the stringent limitations on efficiency of resource usage that the latter entails.

I accept Michael's insight that there must be more to information processing than is embodied in the digital computer as we know it. Clearly, we should be learning and applying all we can from neural, molecular and quantum computing systems. But does structural programmability capture the key distinction to enable us to proceed or is it a manifestation of something else?

3. Limitations of the structural programmability argument

Michael argues that in order for a system to be structurally programmable, it must necessarily use order n of its potential n^2 interactions. This follows because each processor in the system must have a fixed number of connections to others. Were this number to grow as the number of processors increased, then the processor would not be scale invariant, namely, it could not be described once and for all in a user's manual. However, I'm not convinced that the limitation to a fixed number of connections is the germane differentiator between computers and brains. Immediate interaction with a fixed number of neighbors seems to be a characteristic of many physical systems, independently of their status as information processors. Most models of physical systems limit immediate influence to physically adjacent neighborhoods. Indeed, the human brain is notable for manifesting a limitation in connectivity, though not governed by physical proximity, where each of the 10^{10} neurons are synapse-connected to only 10 000 other neurons. And on the other hand, systems not normally considered to be information processors, may be highly interactive. For example, astrophysical systems are claimed to exhibit n-body interactions i.e., where to accurately predict a system's behavior, one has to account for the influence of every galaxy on all the others through gravitation and other forces.

Michael says that giving up on structural programmability allows a system to use all of its n^2 interactions for computation. (He actually reduces this number by half to allow a system to be flexible in evolutionary adaptation.) Although, he never actually asserts that the human brain achieves such efficiency of use, we are left with the suggestion that the brain uses more of its available interactions than a digital computer does of its. This may well be true, but if we go back to the critical distinction, we should ask the question of whether any system can actually use its n^2 interactions and still be bound by the Turing–Church limitations on computability? Unfortunately, the answer is no. In other words, exploiting n^2 interactions is not physically possible according to accepted characterizations of real-world computation.

To see this, consider a finite, but infinitely extendible tape, as typically used in a Turing machine. At any given instant, let the tape have n squares, each with two (local) states. For a tape of length n, there are 2^n global states (or configurations), and no matter what the root cause, we can express the succession from any one state to the next by a transition function, or mapping, which assigns a unique successor to each of the 2^n states. If a Turing machine is the root cause, this mapping will be quite restricted in nature, since a number of constraints, originating from the action of the head, will apply. For example, only one square will be allowed to change at a time (that written by the head), if a square changes in one state, then only it and squares on its left or right, can change in the transition. On the other hand, to say that all n^2 interactions are allowed means that the transition function cannot be expressed in any manner simpler than the following:

In each global transition, all of the squares can change at once and each local transition depends on the states of all the squares. Now, for any given n, 2^n is finite and the transition function is that of a finite automaton. This is clearly simulatable by a Turing machine. However, Michael's programmability argument depends on n being allowed to increase indefinitely. In this case we must allow for a system whose state set is the disjoint union of a each of the n-length subsets, for $n = 1, 2, 3,\ldots$ and whose transition function may specify successors from any one n-length global state to any $n+1$-length global state (assuming that growth is limited to adding a square at the end of the tape). The state set of such a system is countably infinite (the set of all finite sequences of 0's and 1's) but the transition function can be more than a Turing machine can handle. For example, consider a function, f, where $f(x_1, x_2, \ldots, x_n) =$ first $n+1$ bits of a particular non-computable number, for every integer n, and all sequences of length n. Then, started from a one cell square with a zero, the system proceeds to write the non-computable number on the tape. As time progresses, earlier squares remain quiescent and the moving front creates the next square and writes the appropriate bit, thus performing a feat that Turing (1936) showed was not possible for Turing machines.

4. Programmability as supporting an interface

If structural programmability cannot be reasonably identified with a sharp distinction such as exploitation of n^2 interactions, it nevertheless gets at some distinction between computers and brains. What can that be? Let us re-examine the requirements for a system to be programmable from a different perspective. From systems theory, there is a concept of a mathematical system that responds to input time segments by transiting through state trajectories, and producing output time segments (Arbib, 1967; Mesarovic and Takahara, 1975). Such a system can be configured as a computational system if its inputs and output can be suitably interpreted as the interfaces shown in Fig. 1. Stepping back, we note that a system takes on computational properties only in the context of an interaction with a user, interpreted as a programmer. In such an interaction, there must be an input interface that supports:

- inputs interpreted as programs and their data;
- a command to start execution of the current computation,

and an output interface that supports:

- a ready signal indicating that the current computation has terminated;

- a readout that is interpreted as the output computed for the previously given input.

We'll refer to this scheme of input and output interfaces as the *computational interface scheme*. Here, we are using the notion of interface that has become common in object-oriented systems design (Booch, 1991). For example, in client-server systems (Orfali and Harkey, 1997), an interface is an abstraction presented to the client that provides services that can be implemented by a concrete servant object in the server (in a manner transparent to the client). Behind the interface is a contract in which the server promises to deliver an output satisfying certain requirements provided that the client provides certain inputs adhering to certain formats. The programmability interface scheme that we have formulated is quite demanding, and only a relatively small subset of all systems can support such a scheme. Certainly, Turing machines in their usual definitions and conventions do so (program and data descriptions on tapes, start states, halt states, and output tape conventions). Common programming systems and their respective run-time systems taken together also do so. For example, the Java compiler produces bytecode from user input programs and data that can be given to a Java Virtual Machine for execution and output generation.

Now this computational interface scheme can be applied to other-than-digital systems as potential computational systems. For example, a quantum computer has to support such a framework

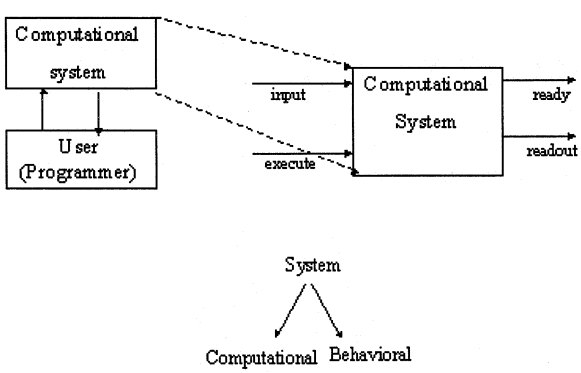

Fig. 1. Formulating a computational system as a general system with a computational interface.

(indeed in this case, indicating termination may be problematic, for example). A system may be engineered from scratch to implement the interfaces or it may be coaxed, cajoled, or forced into it, if that is possible at all. A brain may not be programmable, but ironically, a collection of humans may well be! As recounted in Gigerenzer and Goldstein (1996), before the advent of digital computers, such use of humans as computers was common. For example, groups of women were organized in assembly line fashion to perform complex ballistic table calculations. While any one person could not manage such a computation, it could be correctly implemented by breaking it up into a series of simple arithmetic operations, with results passed from one 'computer' to the next. As another example, Newell, Shaw, and Simon had groups of graduate students collectively hand simulate their novel Logic Theorist before they reduced it to computer code. Searle's Chinese translation room is yet another instance.

Satisfying the computational interface scheme is not a trivial requirement. An interface scheme for structural programmability would extend the basic computational interface scheme since it would demand more visibility, control, and assurance of correctness than would simple programmability. The resources required for implementing such a requirement may be considerable and so reduce those available for other functionality. In this sense, a system harnessed for computation, i.e. constrained to satisfy a structural programmability interface scheme, may be less able to perform other processing functions than in its natural (unharnessed) state. However, this is a different tradeoff than expressed in Michael's original tradeoff principle, namely, that structural programmability entails less efficient use of resources for all forms of computation.

5. What's the right abstraction?

In his 1989 Biosystems article, Michael noted that the McCulloch–Pitts neuron net formalisms showed that it is possible to provide an abstraction of intelligence, not that such an abstraction has to be structurally programmable. Neverthe-

less, the dominant paradigm in cognitive science and artificial intelligence (AI) came to be expressed in Simon's Physical Symbol System Hypothesis (Simon and Newell, 1964) The hypothesis asserts that 'the physical symbol system … has the necessary and sufficient means for general intelligent action.' More explicitly, intelligence is behavior that can be produced by an abstract model capable of certain symbol manipulation operations, independently of how these operations are implemented in real matter. Further the kind of symbol manipulations, copying, storing, retrieving, and comparing, are very much characteristic of certain programming languages such as LISP, the lingua franca of AI. In other words, according to the hypothesis, with enough understanding of how to do it, any intelligent behavior can be expressed in LISP, an abstract model of computation. Apart from performance considerations, nothing about the implementation or execution of the model matters.

Since LISP is an implementation of Church's lambda calculus, and the latter is one of the earliest formalisms to be shown to be Turing equivalent, one reading of the symbol system hypothesis is that intelligence can be captured in any Turing equivalent formalism, again independently of its physical embodiment. Another reading is that the symbolic data structures (lists) and associated operations (list processing) are structurally equivalent, as opposed to merely behaviorally equivalent, to the way intelligent processing occurs in human cognitive systems. In other words, it asserts that human problem-solving strategies are strongly isomorphic to a subset of the symbol manipulation programs of LISP. The problem of building intelligent artefacts reduces to that of finding this particular subset of LISP programs.

Four trends have emerged to undermine the dominance of the Physical Symbol System Hypothesis paradigm:

- Traditional AI programming has not succeeded in demonstrating many kinds of intelligent behavior.
- Revival of the neural net tradition succeeded in demonstrating certain behaviors not achieved with the AI paradigm.
- Doubts arose about the isomorphism between the Symbol System and human intelligent processes (do human information processing work only with list structures and symbol manipulations?).
- Doubts arose about the independence of the Symbol System abstraction from the execution environment (perhaps one needs to understand the interaction between an organism and its environment to understand its brain).

If by digital computer, one means a physical symbol system, then these trends can be seen as confirming Michael's brain-computer disanalogy. Symbol manipulation of the kind that is fundamental to the symbol system hypothesis matches what computers can do. However, it has not found an obvious correlation with processes in the brain, as one might have expected given the increasing ability to observe that activity using PET scans. The very essence of a symbol is that it can be copied at will, moved, and arranged with others in arbitrary sequences. However, as recently admitted by a stalwart of the AI paradigm, William Clancey (Clancey, 1999), there is a yawning gap between such behaviors and what the underlying neural substrate appears to do.

6. Moving to the behavioral frame

As we shall see, in Clancey's view, we have both confirmation for Michael's brain-machine disanalogy and some dissent on the details and on the significance of the disanalogy. However, to better understand these differences, we need to leave the programming framework in which Michael cast the discussion in favor of the behavioral framework, which views organisms in the 'wild' free of the interfaces required to harness their behaviors as computational entities.

6.1. Models of behavior

First, let's see how the computational framework, and the Turing machine, are not the appropriate models for understanding the evolutionary origins and nature of animal and human cognitive processing. Roughly, the standard Turing ma-

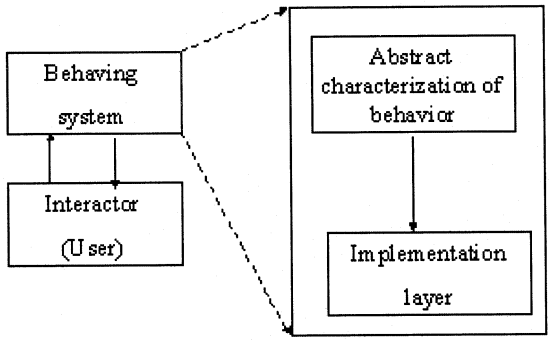

Fig. 2. The Behavior Frame, an alternative to the conventional computational frame.

chine—including any of its information technology equivalents—allows a computation to be any mapping from an input representation to the output representation that results from working in a leisurely manner that we will call the 'deliberative mode'. In this mode, the Turing machine head can scan over the input tape, copy it to its internal memory tape, and revisit and modify this memory any number of times, before finally writing its output on the output tape. In contrast, consider a 'reactive mode', where the head must write its first output after reading the first input, the second output after the second input (without revisiting the earlier input or output), and so on, until the input tape is completely read. With a finite state head, this model is equivalent to a finite state transducer. But now we depart from the Turing machine setting in two ways—(1) the state set of the head is not limited to a discrete finite set; and (2) the operation must be in real time—the head moves forward synchronized with the flow of time and writing on an output no earlier than reading the associated input. In fact, we eliminate the discrete nature of the tape and allow the head to read and write arbitrary values and move in real time. The resulting concept is the input/output causal dynamical system that was mentioned earlier.

Now as in Fig. 2, we specify an appropriate reference frame in which the behaviors of the dynamic system can be interpreted by the interactor (no longer a programmer). Further, we suppose that we can decompose the system into two parts—(1) an abstract characterization of its behaviors in which we and the interactor, are interested; and (2) an implementation system that can generate behaviors. In regard to the class of behaviors, the abstract characterization allows us to compare two different systems for their behavioral equivalence. All we require is that such a class can be formulated, not that we know how to do it at present. This formalization can be instantiated by Turing's test of intelligence, but goes beyond it, in that it allows many other forms of interaction that could elicit behaviors that could be considered intelligent or of interest to the interactor.

Viewed within the behavioral frame and allowing input/output dynamic systems as our class of models, we ask how the abstract behavior characterization and the implementation system relate to each other. A kind of null hypothesis in this regard is that there is *no impact* on the set of intelligent behaviors by the underlying implementation other than pure numbers or speed. This is implicit in Hans Moravec's prediction that by 2050 robot 'brains' based on computers that execute 100 trillion instructions per second (IPS) will start rivaling human intelligence (Moravec, 1999). Underlying this argument, is a behavioral equivalence between neurons in biological brains and IPS in artificial computers. The relevant behavior of each neuron can be replicated by a sufficient number of IPS. It takes so many billions of neurons to create an intelligent human, and likewise, we require so many trillions of IPS to implement an intelligent robot. In a strong form, this equivalence implies that pure brute force can produce intelligence and the structures, neural or artificial, underlying intelligent behavior are of little significance.

6.2. Time-sequenced coordinated conceptual processes

In contradistinction to the null hypothesis, both Michael and Clancey agree that the properties of the implementation system, the bioware, must be taken into account to understand the behaviors well enough to have any chance of their being reproduced by some other implementation system

Table 2
Aspects of (dis)analogy based on implementation properties

Aspect of analogy	Digital computer	Conrad's conception of brain
Mode of operation	String processing mode of operation	Tactile pattern matching at molecular level
Handling of time	Clocking mechanism discretized time: all formal models of computation that are completely general, depend on discretization of time	Time coordination mediated by spatiotemporal pattern recognition capabilities of tactilizing neurons
Knowledge representation	Main information processing at network level: knowledge is in the connections	Main information processing at intraneuronal level; many specialized types of neurons capable of performing a variety of pattern processing tasks
Form of memory	Addressable memory	Memory storage and retrieval probably involves synaptic facilitation and assemblies of neurons
Processing style	Sequential use of resources	Massively parallel
Type of dynamics	Discrete dynamics (simple switches)	Discrete and continuous dynamics mixed
Dependence on implementation substrate	Physical realization of a formal system (equations of physics irrelevant to computing) Symbol hypothesis: independence of intelligent behavior from material realization	Highly dependent on material composition and physical interactions

(e.g. silicon-based computer). Michael claims that the bioware is capable of exhibiting behaviors that cannot be reproduced by programmable computers. This may be the case but we have argued that the programmability issue, per se, is not the heart of the limitation. Table 2 outlines some of the properties that Michael proposed as essential for biological information processing and which make its behavior set, viewed through the appropriate interactor framework, larger than the set capable of being produced by computers. Clancey, while not as elaborate in the properties concerned, focuses on similar properties of bioware, but draws different conclusions. He proposes that these properties *constrain* the behaviors that can be produced. More specifically, the primitive behaviors of the Physical Symbol System cannot be realized in representations in which symbols are mapped in one-to-one correspondence with neuron assemblies and time is taken into account (Clancey, 1999). These implementation features greatly constrain the processing that can be done. Noting that assemblies that are once-only embodiments of concepts can only be re-activated after some refractory period, Clancey shows that such processing limitations correlate with observations that have been made in the linguistics of complex sentence recognition. For example, certain grammatical sentences with recurring embedded phrases are easy for humans to make sense of, while others with similar formal structure, are hard to understand, or even nonsensical. The explanation lies in the fact that the unacceptable sentences may require immediate reactivation of phrase structures with the same referent, a requirement that can not be implemented with assemblies that need to rest for a significant period before becoming reactivatable. As a consequence, brains can not produce certain behaviors that symbol systems implemented in silicon can—indeed, it is in such constrained behaviors that we can find clues about how the implementation system works.

6.3. Fast and frugal faculties

So far, we have introduced time-boundedness and properties of the bioware implementation as essential to the understanding of intelligent biological behavior. These factors become part of a

larger framework when we move to an evolutionary perspective. In this perspective, the real world is a threatening environment where knowledge is limited, computational resources are bounded, and there is no time for sophisticated reasoning. Unfortunately, traditional models in cognitive science, economics, and animal behavior have used theoretical frameworks that endow rational agents with full information of their environments, unlimited powers of reasoning, and endless time to make decisions. Our understanding of computer-brain disanalogy requires first an understanding of the essence of intelligence as a problem-solving mechanism dedicated to the life and death survival of organisms in the real world. Evidence and theory from disparate sources have been accumulating that offer alternatives to the traditional paradigm.

An important crystallization of the new thinking is the 'fast, frugal, and accurate' (FFA) perspective on real-world intelligence promoted by Todd and Gigerenzer (Gigerenzer and Todd, 1999; Gigerenzer and Goldstein, 2000). FFA heuristics are simple rules demanding realistic mental resources that enable both living organisms and artificial systems to make smart choices quickly with a minimum of information. They are accurate because they exploit the way that information is structured in the particular environments in which they operate. Todd and Gigerenzer show how simple building blocks that control attention to informative cues,

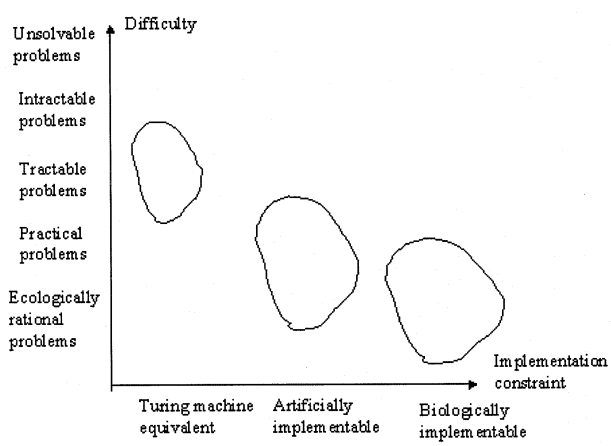

Fig. 3. Relation between problem difficulty and the underlying implementation environment.

terminate search processing, and make final decisions can be put together to form classes of heuristics that have been shown in many studies to perform at least as well as more complex information-hungry algorithms. Moreover, such FFA heuristics are more robust than others when generalizing to new data since they require fewer parameters to identify.

6.4. Exploiting ecologically rational environments

Fig. 3 provides a framework for understanding the place of FFAs within current conceptions of computational complexity. A stratification of problem classes ordered by decreasing difficulty is:

- Unsolvable problems: problems for which no equivalent Turing machine can exist. For example, the halting problem (deciding after a finite analysis whether a given Turing machine will eventually halt) is unsolvable.
- Intractable problems: nondeterministic polynomial (NP)-complete, unless P (below) $= NP$, these represent problems whose solution time grows exponentially with their size. Besides such famous problems as the travelling salesman problem, essential tasks in vision, reasoning, and behavior of artificial neural nets are known to be NP-complete (Tsotsos, 1991).
- Tractable problems: problems that can be solved in a time that is proportional to a polynomial in the size parameter (P).
- Practical problems: problems that can be solved in a time that is slowly growing in the size parameter, or are small enough to be solved in acceptable time (even though they would grow at least as polynomial with size).
- Ecologically Rational problems: problems that are formulated within their real-world context.

Recent results in probabilistic computation by artificial neural networks (Hangartner and Cull, 2000) are consistent with the conclusion that within the framework of traditional computation, neural networks do not exceed the abilities of standard programmable models in relation to tractable problems but may provide more efficient use of resources. Thus, there is little hope that biological systems, or biologically-inspired implementations, will have any greater computing power than al-

lowed by the Turing–Church hypothesis. So how do biological organisms solve the basic problems of survival in a hostile world?

Fig. 3 illustrates the answer underlying the fast and frugal paradigm. Biologically implementable behaviors are tuned to the environments in which organisms are situated. Fast and frugal heuristics work well with ecologically rational problem data sets that are extracted from real-world situations. Thus, FFAs are a different breed of heuristics. They are not optimization algorithms that have been modified to run under computational resource constraints, e.g. tree searches that are cut short when time or memory run out. Typical FFA schemes enable ignorance-based and one-reason decision making for choice, elimination models for categorization, and satisfying heuristics for sequential search. Leaving a full discussion of the differences to Gigerenzer and Todd (1999) the critical distinction is that FFA's are structured from the start to exploit ecologically rationality. In other words, they make restrictive assumptions about their input data, such as that they are characterized by J-shaped frequency distributions. They work well because these assumptions often happen to hold for data from the real world. Thus, FFAs are not generic inference engines operating on specialized knowledge bases (the paradigm of expert systems) nor other generalized processing structures operating under limited time and memory constraints. An organism's FFAs are essentially *reflections* of the real environment in which it has found its niche and to which it has (been) adapted.

7. The role of attention

'Fast and frugal' operates at the cognitive level where sequential processing appears to predominate. At the level of perception, processing is orders of magnitude finer grained and highly parallel. The question then is whether ecological rationality also operates at this level. Pinker (1997) presents evidence that our visual systems are highly tuned to work within the texture of scenes that are consistent with savanna landscapes, characteristic of the environments of our evolutionary ancestors. However, this may not be the whole story since even under such constraints, the problems of visual processing are inherently intractable in the technical sense defined earlier. Tsotsos (1995) shows that an attentional selection mechanism is needed to deal with such intractability. For example, the problem of visual search where targets are not given in advance is NP-complete. However, if targets are given the problem becomes linear in the size of the display, i.e. very practical (Tsotsos, 1993, 1995). Attentional selection must be implemented as a mechanism of practical complexity. It may focus processing on the mapping to try first, i.e., provide priorities to the processing tasks, where each processing stage is practical. Of course, attentional selection cannot be foolproof—there can be no guarantee of its validity in general, since otherwise the overall visual processing would become practical, rather than NP-complete.

We conclude that ecological rationality, fast and frugal heuristics at the cognitive level of processing, and attention-directed perceptual processing are keys to understanding the characteristics of biological information processing. We need to understand these better, and other constraints like them, to enable artificial systems to have life-like intelligence[1].

[1] The question is raised as to whether the fast and frugal paradigm has thermodynamical correlates. Pinker (1997) provides our basic perspective. Any natural or artificial system must live within the constraints imposed by the underlying material substrate on its information processing. Information has costs: (a) space, the hardware to hold it; (b) time, life is a series of deadlines. If too much information is accessible, processing it may take too much time. A system must put survival before sagacity; (c) resources, information processing and storage require energy, which is always in limited supply. The implication is well stated by Pinker: "any intelligent agent incarnated in matter, working in real time, and subject to the laws of thermodynamics must be restricted in its access to information. Only the information that is relevant should be allowed in. This does not mean that the agent should wear blinkers or become an amnesiac. Information that is irrelevant at one time for one purpose might be relevant at another time for another purpose. So information must be *routed*. Information that is always irrelevant to a kind of computation should be permanently sealed off from it. Information that is sometimes relevant and sometimes irrelevant should be accessible to a computation when it is relevant, in so far as that it can be predicted in advance" (Pinker, 1997). From this perspective, fast and frugal processing is one means of living within the thermodynamical constraints imposed on all systems.

8. The discrete event abstraction

New kinds of models for biological neurons provide possible mechanisms for implementing intelligence that is characterized by fast and frugal heuristics in ecologically rational environments. Work by Gautrais and Thorpe (1998) has yielded a strong argument for 'one spike per neuron' processing in biological brains. 'One-spike-per-neuron' refers to information transmission from neuron to neuron by single pulses (spikes) rather than pulse trains or firing frequencies. A face-recognition multilayered neural architecture based on the one-spike, discrete event principles has been demonstrated to better conform to the known time response constraints of human processing and also to execute computationally much faster than a comparable conventional artificial neural net (Rullen et al., 1998)[2]. The distinguishing feature of the one-spike neural architecture is that it relies on a temporal, rather than a firing rate, code for propagating information through neural processing layers. This means that an interneuron fires as soon as it has accumulated sufficient input 'evidence' and, therefore, the elapsed time to its first output spike codes the strength of this evidence. In contrast to conventional synchronously timed nets, in fast neural architectures single spike information pulses are able to traverse a multi-layered hierarchy asynchronously and as fast as the evidential support allows. Thorpe's research team has also shown that 'act-as-soon-as-evidence-permits' behavior can be implemented by 'order-of-arrival' neurons, which have plausible real world implementations. Such processing is invariant with respect to input intensity because response latencies are uniformly affected by such changes. Moreover, coding that exploits the firing order of neurons is much more efficient than a firing-rate code, which is based on neuron counts.

In recent work, Zeigler (2001) shows how the discrete event abstraction captures information in rich continuous data streams in terms of events and their timed occurrences (Zeigler et al., 2000). The evidence from recently developed models of neurons and neural processing architectures and from fast and frugal heuristics provide further support for the centrality of discrete event abstraction in modeling reactive behavior when the constraints of space, energy and time are taken into account. There has been increasing interest in how time enters into neural processing in brains. With recent experimental work summarized by Singer (1999), one major question being asked is whether or not synchronization among cell firings plays a major role in fusing the results of lower-level processing into higher-level elements for further processing.

8.1. The synchronization problem

Although creating discrete event simulation models that work on dynamic binding principles seems quite challenging, we suggest that using discrete event modeling would provide the right level of abstraction to address these challenges. One problem in neural nets using strength-to-latency coding that emerges immediately is that processing is fundamentally asynchronous. Unlike in conventional models such as cellular automata, there is no global clock tick to synchronize states of neurons at a recurring time step. Consequently, as illustrated in Fig. 4, laggard spikes from earlier percepts may interfere with spikes stimulated by later percepts. One solution is to place a time-out on the response of a neuron—it is reset to its initial state after a given duration. However, it may be difficult to retain proper phasing over time in this self-reset approach. Another solution is to provide a synchronizing pulse that resets all neurons at the end of each percept. This represents only a partial step toward full global cellu-

[2] The face-recognition layered net was executed by a discrete event simulator and took between 1 and 6 s to recognize a face on a Pentium PC vs. several minutes for a conventional net on a SGI Indigo. Recognition performance in both cases was very high. The authors employed a training procedure that while effective, is not plausible as an in-situ learning mechanism.

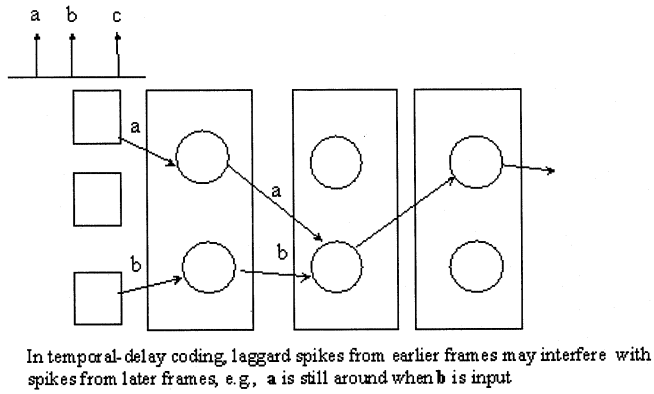

Fig. 4. Laggard pulses in strength-to-latency information transmission.

lar-automata-like regulation since within the intervals between percepts, neurons can respond asynchronously. Global timing of this nature is hypothesized as fundamental in MacKay's node structure theory, with considerable empirical backing (MacKay, 1987).

Another direction pointing toward discrete event neurons is suggested by the recent interest in the computational capabilities of 'pulsed neural networks' (Maas and Bishop, 1999). Interestingly, research reported in this direction, in the main, does not take the full step toward discrete event abstraction. Such abstraction offers a simplification of concept that then stimulates further advances in the new space opened up by the simplified conceptual framework. For example, adoption of switching (Boolean, combinatorial) logic allowed design of digital computers and, in freeing the abstraction from its technological embodiment, gave birth to a completely new field of computer science. In the case of switching logic, the technological embodiment had to be in place to ensure the stability of the two distinct states representing the abstractions 0 and 1 and the transitions specified by designers. Discrete event abstracted neurons suffice to provide shortest path solutions to directed graphs. This suggests an evolutionary explanation for the structure and behavior of biological neurons. If they operate on discrete event principles, they are optimized for operation in a world requiring synthesis of quick multistep reactions as dictated by shortest paths in graphs of dependent actions.

9. Conclusions

Table 3 summarizes the aspects of the brain-machine disanalogy as viewed from the perspective of discrete event abstraction as the means to implement fast and frugal heuristics and the characteristics of biological information processing.

Although, time-boundedness of real-time decision making and temporal organization of perception and action are intrinsic to intelligent behavior, few temporal neuro-computational models have been proposed (Cariani, 2001). Discrete event abstractions may help develop such models and suggest some ways in which state-of-the-art computers do not work as humans do. A few are summarized under the rubrics:

- *What comes to mind*, our immediate responses to real-world problems pop out without much thought, most options don't come to mind and implausible options do not occur to us. This is compatible with networks of discrete event neurons implementing first-arrival-takes-all so that the fastest response wins.
- *On second thought*, we can censor or revisit choices made on first blush with further analysis. Discrete event neural networks can support generation of multiple responses that can range from quick and dirty with low priority to slow and deliberative with high priority. Deadlines and priorities then determine which responses are taken. In other words, a second thought, or reconsideration, may be the result of a slower,

Table 3
Aspects of brain-machine (dis) analogy from the viewpoint of discrete event abstraction

Aspect of analogy	Brain as an implementation of discrete event abstraction
Activity	Agents may be proactive or reactive
Exploitation of interactions	Interactions are prioritized in the service of current goals
Structural programmability	Efficiency achieved through exploitation of race analogy in temporal dimension
Mode of operation	Deliberative vs. reactive mode of processing
Handling of time	Time is not discretized as a continuous quantity, it plays a central role in processing
Knowledge representation	Connections carry associative knowledge as well as delays encoding importance of evidence in race competitions
Form of memory	Concepts mapped one-to-one to neuron assemblies
Processing style	Distributed, asynchronous, fast and frugal, attention-mediated, tuned to ecological rationality
Type of Dynamics	Implementation of discrete event abstraction
Dependence on implementation substrate	Discrete event abstraction middle ground—implemented by physical layer and tied to the physical nature of time

more deliberative process that wins out over a faster one because it has higher priority and finishes before the deadline.

- *Tip of the tongue*, we can have the sensation of knowing that we know the answer without having it in mind. Discrete event neural networks can overlay the multiple response net to monitor for the results of competing processes as they complete. The priorities of the faster responses provide an estimate of the confidence in the available answer. An early response with low priority would generate an estimate of low confidence, while a fast response with high priority would generate a confident response.

These observations open up research directions centered upon discrete event simulation models of bioware-implementated behaviors. They require bringing together multidisciplinary teams including such subject matter experts as computer scientists and engineers, cognitive scientists, neuroscientists, and evolutionary biologists. The progress made would be a better understanding of information processing in biological organisms and how these insights might be transferred to the design of artificial brains capable of acting in the real world.

Acknowledgements

This research is partially supported by NSF grants 'Next Generation Software', EIA-9975050, and 'DEVS Formal Framework for Scalable Enterprise Systems', DMI-0075557.

References

Arbib, M.A., 1967. Theories of Abstract Automata. Prentice-Hall, Englewood Cliffs, NJ.

Booch, G., 1991. Object-Oriented Design with Applications. Benjamin/Cummings, Redwood City, CA.

Cariani, P., 2001. Systems and Dynamics in the Brain. Biosystems 60 (1–2), 59–83.

Clancey, W.J., 1999. Conceptual Coordination: How the Mind Orders Experience in Time. Lawrence Erlbaum s Associates, Mahwah, NJ.

Conrad, M., 1988. The price of programmability. In: Herken, R. (Ed.), The Universial Turing Machine. Oxford University Press, Oxford, UK, pp. 285–308.

Conrad, M., 1989. The Brain-Machine Disanalogy. Biosystems 22, 197–213.

Gautrais, J., Thorpe, S., 1998. Rate coding versus temporal coding: a theoretical approach. BioSystems 48 (1–3), 57–65.

Gigerenzer, G., Goldstein, D.G., 1996. Mind as computer: birth of a metaphor. Creativity Res. J. 9 (2 and 3), 131–144.

Gigerenzer, G., Todd, P.M., 1999. Simple Heuristics That Make Us Smart. Oxford University Press, Oxford, UK.

Gigerenzer, G., Goldstein, D.G., 2000. Reasoning the fast and frugal way: models of bounded rationality. In: Connolly, T. (Ed.), Judgment and Decision Making. Cambridge University Press, Cambridge, pp. 621–650.

Hangartner, R.D., Cull, P., 2000. Probabilistic Computation by Neuromine Networks. Biosystems 58 (1–3), 167–176.

Maas, W., Bishop, C.M., 1999. Pulsed Neural Networks. MIT Press, Cambridge, MA.

MacKay, D.G., 1987. The Organization of Perception and Action. Springer, New York.

Mesarovic, M.D., Takahara, Y., 1975. General Systems Theory: Mathematical Foundations. Academic Press, New York, NY.

Moravec, H., 1999. Rise of the Robots. Scientific American (August), 124–132.

Orfali, R., Harkey, D., 1997. Client/Server Programming with Java and CORBA. Wiley Computer Publishing, New York, NY.

Pinker, S., 1997. How the Mind Works. W.W. Norton, New York, NY.

Rullen, R.V., Gautrais, J., et al., 1998. Face processing using one spike per neurone. BioSystems 48 (1–3), 229–239.

Simon, H.A., Newell, A., 1964. Information processing in computer and man. Am. Scientist 52, 281–300.

Singer, W., 1999. Neural synchrony: a versatile code for the definition of relations. Neuron 23, 49–65.

Tsotsos, J.K., 1991. Computational resources do constrain behavior. Behav. Brain Sci. 14, 506–507.

Tsotsos, J.K., 1993. An inhibitory beam for attentional selection. In: Harris, L., Jenkin, M. (Eds.), Spatial Vision in Humans and Robots. Cambridge University Press, Cambridge, pp. 313–332.

Tsotsos, J.K., 1995. Modeling visual attention via selective tuning. Artificial Intelligence 78, 507–545.

Turing, A.M., 1936. On computable numbers, with an application to the Entscheidungs problem. Proc. London Math. Soc. 42, 230–265.

Zeigler, B.P., 2001. Discrete event abstraction: an emerging paradigm for modeling complex adaptive systems. Festschrift in honor of John H. Holland. L. Booker.

Zeigler, B.P., Kim, T.G., et al., 2000. Theory of Modeling and Simulation. Academic Press, New York, NY.