On the Role of Randomization in Minimizing Neural Entropy

Stuart H. Rubin
SPAWARS SYSTEMS CENTER
53560 Hull St.
San Diego, CA 92152-5001, USA
E-mail: srubin@spawar.navy.mil

and

Ljiljana Trajkovic
Simon Fraser University
Vancouver, BC Canada
E-mail: ljilja@cs.sfu.ca

ABSTRACT

We consider the scruffy vs. the neat approach as first proposed by Minsky. In particular, the brain has been modeled as a society of mind. At some levels, the brain clearly processes information, at least in part, using a scruffy approach (e.g., vision). At other levels, the brain clearly processes information, at least in part, using a neat approach (e.g., logical reasoning). It has been proven by Lin and Vitter that neural networks having a hidden layer are NP-hard to train. Thus, a domain-specific representation is needed for at least purposes of scalability. This paper asks and attempts to answer the question, "Does the brain translate neural data into a symbolic representation for purposes of concept formation and creative thought?" Conversely, "Can a society of neural networks learn to structure the data so as to decrease its entropy?" If so, then the meaning of a symbol cannot be truly had from its textual form, but rather from its relation to other symbols.

Keywords: Associative Mining, Case Association, Fuzzy Mapping, Fuzzy Mining, Logically Associative Memory, Randomization, Veristic Calculus

1. INTRODUCTION

Method

We acknowledge the fundamental need for domain-specific knowledge in keeping with Rubin's proof of the Unsolvability of the Randomization Problem [1]. The question as to the degree to which a neural net, or system of networks, can learn to decrease its entropy may be reduced to the capability for forming domain-specific generalizations. This capability for domain-specific generalization goes well beyond the statistical boundaries of contemporary neural net generalization. A second application of the Unsolvability of the Randomization Problem reduces the capability for forming domain-specific generalizations to the capability for forming domain-specific representations. But then, this capability is dependent on previous structure. Thus, it is argued that the entire issue of the neat vs. scruffy approach is provably irrelevant. Instead, the brain must form domain-specific representations using evolutionary randomization (see below). All that matters is to minimize the entropy of the information-theoretic model using randomization. Of course, there is also a practical dimension that involves issues pertaining to spatial-temporal realization. However, even here the Unsolvability of the Randomization Problem implies the necessary search for ever-better domain-specific hardware.

Results

We know that the central reason for the failure of the Japanese Fifth Generation project [2] is that their system was built to be capable of deductive, but not inductive reasoning. In other words, it followed a second order predicate calculus model. They learned that the logic must yield to the algorithm in all matters pertaining to creative or inductive reasoning. The quality of an algorithm, in turn, hinges on the degree to which its space-time image can be minimized through reuse. We note that any algorithm sufficiently complex to be capable of self-reference can never be certified as bug free. This is a law of nature. Nevertheless, testing and scalability are maximized if reuse is maximized. Construction time is minimized if reuse is maximized. RISC chips are but one example of these principles applied to hardware design. Simply put, creativity is necessarily of an algorithmic nature and non-trivial algorithms and their associated platforms (i.e., of significant scale) cannot be had or run in the absence of first principles of randomization and reuse. If all this sounds a little foreign, it may help to recall that relativistic effects may also sound foreign to the average person due to the relatively high velocity of light with respect to their daily experiences.

New or Breakthrough Aspects of Work

It has been shown (e.g., the Wizard neural system) that the triangle inequality applies to neural systems. For example, if a neural system may learn to recognize a fork invariant of its position using c fundamental memories. However, if two neural systems are constructed and connected such that the first learns to rotate an object such as a knife to a normal position and the second then attempts to recognize this object, then the total number of fundamental memories needed to recognize the fork invariant of its position will be significantly less than c (i.e., the triangle inequality). The challenge is to scale such problem reductions. It is argued that an object-oriented society of mind model will serve to facilitate analysis. For example, if one neural network is trained on what a car is and a companion network is trained say on what a Toyota Corolla is (i.e., an instance of car), then not only is the entropy of the system decreased, but the reusability of the networks is proportionately increased. Fig. 1 shows how it is that fuzzy rules can reduce the cost of knowledge acquisition in expert systems (e.g., kasers).

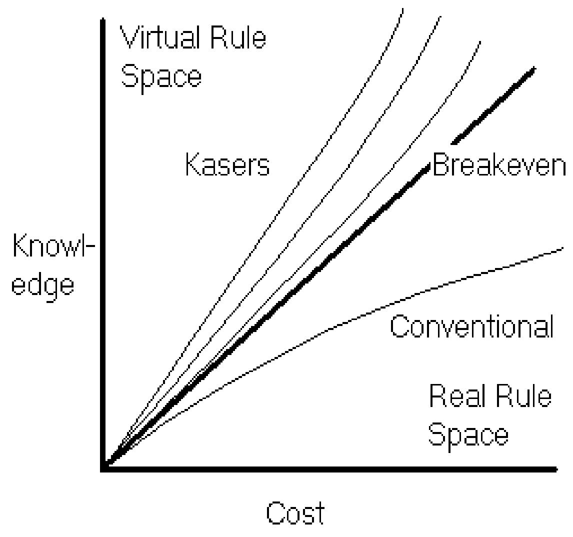

Fig. 1 The Comparative Costs of Knowledge Acquisition

2. RANDOMIZATION

Randomization is a term first coined by Gregory C. Chaitin in a 1975 Scientific American article entitled, "Randomness and mathematical proof" [3]. This concept explains why certain sequences of numbers are said to be random – i.e., because a generator, which is more compact than the sequence itself cannot be found. Thus, if one writes the sequence: 0101010101010101…, then you see a pattern and conceive an algorithm that is a compaction (randomization) of that sequence. Here, the sequence is clearly not random. This concept shows how the entropy of a software system is decreased if data is represented as a program wherever possible (e.g., Euler equations in lieu of tables of logarithms) [3]. If this is not possible, then the data is already randomized.

Knowledge

We tend to think of knowledge as being a collection of facts. However, randomization theory informs us that this is a misnomer [4]. A collection of facts is properly termed information. The relative maximal compression of the collection of facts leads to knowledge because such compressions necessarily lead to a capability for commonsense reasoning (see below) as well as an explanatory capability. In other words, one knowledge source can be said to be more informative than another if the density of information that it encodes is greater per measured unit.

3. THE TYPE II KASER

The Type II kaser is a knowledge amplifier that differs from a Type I kaser in that it automatically evolves a structural declarative hierarchy of predicates [5]. Moreover, it uses a context-free grammar to randomize the rule base and in so-doing facilitates rule selection and specification.

Using conventional expert systems, excessive rule complexity practically forces one to resort to case-based reasoning (CBR). The problem with using CBR is that it is not practical to automatically generalize cases, which tends to delimit the utility of the overall system. The use of a hierarchical predicate representational formalism circumvents this difficulty by allowing arbitrarily complex rules to be compactly represented.

Production rules are universal. Nevertheless, one early concern arose in their use. The problem was how to represent sequence. For example, in a diagnostic expert system, one needs to explicitly specify what to do next on the basis of the results of previous tests. A blackboard approach was taken in keeping with the implied goal of randomization (i.e., maximizing reuse throughout the system). Special primitives allow posting and retracting predicates to and from the blackboard respectively. This is a very powerful mechanism for maximizing reuse because the blackboard maintains a hierarchical context.

The topics of truth maintenance (i.e., retracting rules that are no longer valid) and incremental algorithms are not addressed here. Here, we will endeavor to show how knowledge amplification can be accomplished by a Type II kaser using numeric, phrase, or computed predicates. In particular, the Type II kaser defines a paradigm for computing with words [1], [6].

Declarative Tree Acquisition

Consider the inductive acquisition of a new or modified rule by the kaser. The following two cases will be seen to cover all of the possible interactions. Notice that the two cases are symmetric.

Case 1:

$$\begin{matrix} a_1, a_2, \ldots, a_i, \ldots, a_{n-1}, a_n \rightarrow c \\ a_1, a_2, \ldots, a_j, \ldots, a_{n-1}, a_n \rightarrow c \end{matrix}, \; a_i \neq a_j$$

Here, c represents a consequent sequence. In this case, a generalization of a_i and a_j is formed:

$$A_u \rightarrow a_i \mid a_j$$

The user is queried to supply semantics for A_u, or the non-terminal is allowed to remain undefined. Observe that if a_i is empty or absent, then the containing rule is more general than the rule containing a_j, which it replaces. In the present case, the pair of rules is replaced by the generalized rule:

$$a_1, a_2, \ldots, A_u, \ldots, a_{n-1}, a_n \rightarrow c$$

No new rules will be induced by the immediate substitution using the generalized rule. If a_j were replaced by A_u, then a_i would simply become another disjunct in its expanded definition.

Note that distinct antecedent predicates in distinct rules are not matched just in case one is a specialization of the other. For example, “Ford” and “Automobile” do not match because while both will run on high-test gas, not all automobiles will, like a Ford, run on low-test gas.

Case 2:

$$\begin{array}{l} a \rightarrow c_i \\ a \rightarrow c_j \end{array}, \; c_i \neq c_j$$

Here, a represents an antecedent set. In this case, a generalization of c_i and c_j is formed:

$$C_u \rightarrow c_i \mid c_j$$

The user is queried to supply semantics for C_u, or the non-terminal is allowed to remain undefined. Observe that the resolution between c_i and c_j requires additional context. C_u is posted to the blackboard and the additional context allows for the reuse of this conclusion (i.e., by whatever path it was derived). In the present case, the pair of rules is replaced by the generalized rule:

$$a \rightarrow C_u$$

No new rules will be induced by the immediate substitution using the generalized rule. If c_j were replaced by C_u, then c_i would simply become another disjunct in its expanded definition. Consequent predicates are represented as singletons, since they are immutable.

Numeric Classes
Antecedent and consequent classes may be treated the same when it comes to numeric classing. For example,

$$[0,10] \rightarrow [0,4] \mid [5,10]$$

$$[0,10] \rightarrow [0,9] \mid [1,10]$$

$$[0,10] \rightarrow [0,0] \mid [1,9] \mid [10,10]$$

Creative Contextual Matching
Rules are acquired in exactly four situations. First, they are acquired whenever the rulebase is empty. Second, they are acquired to correct an erroneous action. They are acquired upon the validation of the action produced by a generalized antecedent predicate(s). Finally, they are acquired upon the exercise of a timeout or manual interrupt.

It is assumed that all specializations of a predicate are valid. That is, a class may be substituted for by any of its instances. For example, if a context contains “Ford” and a rule antecedent contains “automobile”, then the antecedent predicate can be matched by specializing it. We say that the rule is more general than the context in this instance. Here are the possible cases:

Case 1:
The context covers some rule antecedent.
In this case, the most-specific covered rule is the one selected for firing.

Case 2:
A specialization of some rule antecedent results in a covering by the context.
In this case, the most-specific covered rule is the one selected for firing.

Case 3:
A generalization of some rule antecedent results in a covering by the context.
In this case, the least generalized rule antecedent that is covered by the context is fired. The rule is updated upon validation where generalized predicates replace their specializations.

Case 4:
A generalization/specialization of some rule antecedent results in a covering by the context.
In this case, the least generalized rule antecedent that is covered by the context is fired. The rule is updated upon validation where generalized predicates replace their specializations.

Case 5:
The context cannot cover any rule antecedent using specialization and generalization or the associated action needs updating.
In this case, the user is queried for an appropriate action and the rule is acquired/replaced.

Grammatical Randomization
The antecedent and consequent predicate hierarchies described above are maintained in two context-free predicate grammars (CFGs). These grammars find use in context-matching operations.

Another pair of CFGs are maintained to provide assistance in randomizing across contextual and procedural predicates. These concept grammars work in conjunction with the predicate grammars. An example will serve to make clear their respective roles.

Consider the domain of weather forecasting and the following three normalized rules.
Merge concept and predicate grammars?

$$A_1 \rightarrow C_1$$

$$A_2 \rightarrow C_2$$

$$A_3 \rightarrow C_3$$

The associated antecedent concept set grammar is:

$\mathrm{A}_1 \rightarrow$ Cold, Bad Weather

$\mathrm{A}_2 \rightarrow$ Warm, Bad Weather

$\mathrm{A}_3 \rightarrow$ Light Wind, Good Weather

Bad Weather $\rightarrow$ Cloudy, Falling Barometer

Good Weather $\rightarrow$ Fair, Rising Barometer

The associated consequent concept sequence grammar is:

$C_1 \rightarrow$ Snow, Sleet, Hail

$C_2 \rightarrow$ Rain, High Humidity

$C_3 \rightarrow$ Sunny

The associated antecedent predicate set grammar is:

Cold $\rightarrow$ Below 32° | Frosty Windows

Warm $\rightarrow$ Above 32° | Puddles

Cloudy $\rightarrow$ Cirrus | Cumulus | Stratus

The associated consequent predicate sequence grammar is:

Snow $\rightarrow$ Large Flakes | Small Flakes

High Humidity $\rightarrow$ Foggy | Condensation

Suppose now that the user were to enter that it is 10 degrees F outside. The system would automatically generalize this predicate to Cold using the antecedent predicate set grammar unless more than one generalization could be had – in which case the user would be queried. The operation here is that of associative memory. Alternatively, if no associated semantics had been given (e.g., X001), then the system would ask the user to confirm Frosty Windows if not on the blackboard given more than one possible generalization. The user could also enter a random choice (i.e., a new term to be acquired by the system for this or any other definition). Here, we may assume that the user confirms that it is Cold.

Next, the system references the antecedent concept set grammar to ask the user if there is Bad Weather if not on the blackboard. Note that if the predicate, "Cold" were present in more than one set, the system would offer more than one choice – including the random one as previously described. Here, the user has the option of expanding Bad Weather to find that the necessary precondition for Bad Weather is that the barometer be falling and that it be cloudy. The definition of cloudy can be expanded for the user by the antecedent predicate set grammar – again if not on the blackboard.

Confirmation here leads to the A_1 reduction, which leads to the firing of the first rule – producing action, C_1. The consequent concept sequence grammar predicts Snow, Sleet, and Hail in that order. The type of Snow is determined through the consequent predicate sequence grammar. The use of the blackboard and/or a query will allow the system to determine the granularity of the snowfall (e.g., if snow is beginning, then the flakes will be large and vice versa). This is possible because Snow is written to the blackboard.

A key issue pertains to the method used to randomize any of the grammars. Randomization not only saves space, but more importantly associatively extends what a user is attempting to communicate to the system, or in the case of the consequent grammars, what the system is attempting to communicate to the user. Randomization of the predicate grammars has been defined above. The concept grammars are not to be theoretically randomized by seeking the optimal randomization. This is because as the name implies future randomizations will be dependent upon the identification of concepts that in turn are defined by the left and right hand sides of the rules as they are acquired. We see that as is the case with humans, learning in a Type II kaser is very much sequence dependent.

Fuzzy Programming

Unlike the case for conventional expert systems, a kaser cannot be used to backtrack consequents (i.e., goal states) to find multiple candidate antecedents (i.e., start states). The problem is that the preimage of a typical goal state cannot be effectively constrained (i.e., other than for the case where the general and specific stochastics are both zero) in as much as the system is qualitatively fuzzy. Our answer is to use fuzzy programming in the forward-chained solution. This best allows the user to enter the constraint knowledge that he/she has into the search. For example, if the antecedent menus are used to specify 'CAR' and 'FUEL' for the context and the consequent is left unconstrained for the moment, then the system will search through all instances, if any, of CAR crossed with all instances of FUEL (i.e., to some limiting depth) to yield a list of fully expanded consequents. Generalization-induced system queries, or consequents that pose questions, if any, will need to be answered to enable this process to proceed. Thus, in view of the large number of contexts that are likely to be generated, all interactive learning mechanisms should be disabled or bypassed whenever fuzzy programming is used. Note that CAR and FUEL are themselves included in the search. Each predicate can also be instantiated as the empty predicate in the case of the antecedent menus, if user-enabled. If the only match occurs for the case of zero conjuncts, then the consequent tree is necessarily empty. A method for fuzzy programming, is to simply allow the user to split each conjunct into a set of disjuncts and expand all combinations of these to some fixed depth to obtain a list of contexts. This use of a keyword filter, described below, is optional. For example, the specification $(A \vee A' \vee !A'') \wedge (B \vee B') \wedge (C)$ yields 23 candidate contexts – including the empty predicate (i.e., if one assumes that A'' is primitive and allows for redundancy), which excludes the empty context. The exclamation mark, "!" directs the system to expand the non-terminal that follows it to include (i.e., in addition to itself) all of the next-level instances of its class. For example, !CAR would yield (CAR TOYOTA FORD MAZDA HONDA ... λ). Here, lambda denotes the empty predicate and is included as a user option.

4. ASSOCIATIVE MINING

Fuzzy Mining

The kasers will mine rules from basis knowledge in much the same manner as people use to induce and deduce new knowledge. However, they are not designed to mine deep knowledge – the type of knowledge that lies beyond the capability of the human cognition to discover without thought, reflection, and the application of a great deal of commonsense and domain-specific knowledge. For example, it is not trivial to discover Keppler's laws of planetary motion using the spatial coordinates of the planets over some time frame. Nevetheless, these laws could be discovered using fuzzy mining.

The architecture of a fuzzy mining system may be described as follows. There are seven fundamental components:

1. A domain-specific language that facilitates the expression of alternatives program statements. For example:
   IF x<y then call A or B or C;
   At one extreme this interface can be a natural language or even entirely graphically based – depending upon the application domain.

2. An inference engine that allows instances of the fuzzy program to be distributed for evaluation (e.g., across parallel hardware).
3. An evaluation engine that prunes testing an instance whenever the probability of success based on previous tests does not warrant it. This step is necessary to maximize coverage of the candidate instance space.
4. An optimizer that applies acquired (domain-specific) transforms to reduce the complexity of the result.
5. A feedback mechanism to enable the iterative specification of a fuzzy program in step (1). This is necessary because if too much burden is placed on the computer, the results will prove to be intractable. Conversely, if too much burden is placed on the user, the results again will prove to be intractable.
6. An inference engine that allows the discovered rules to be applied as in a conventional expert system.
7. An explanation subsystem that can convert any discovered rules back into a higher-level user explanation.

Fuzzy miners can operate with unstructured data too. For example, text can be mined for the occurrence of words or phrases in sequence, words or phrases categorized by hierarchical relationships, as well as by schema-driven relations.

Fuzzy miners can operate on models too. For example, the precise causality of a model response may not be known, but it may be localizable to a limited number of alternatives – especially in any hierarchical model. Data concerning the model's behaviour may then be used to instantiate the model and define it. Indeed, multiple data sets may induce distinct models.

Fuzzy mining is not without practical challenges. These challenges include the following issues. First, how can evolution be set up to insure quadratic convergence? Then, if real exponents, roots, etc. are included how can they be discovered in a multivariate context? Clearly, this is not always possible. Also, the reliance upon domain-specific functions and knowledge implies a high degree or retooling from domain to domain. Finally, there is an implied need for *change detectors* – that is, how do we know when the database represents a sub-problem if not an entirely distinct problem? These concerns all boil down to a heavy reliance on domain-specific knowledge. Thus, kaser research may be fundamental to the solution of fuzzy mining problems and thus problems in evolutionary programming in general [7] [8].

Fuzzy Mapping

Fuzzy mapping is an extension of fuzzy programming. Here, the task is to map a feature-enhanced input vector or matrix with one or more dependent categorizations. The mapping function is a fuzzy program. Such work has been implemented using neural networks [9], but while nets having a hidden layer can learn, the learning is critically dependent on the input representation. In other words, in the absence of a good representation, the learning has been proven to be NP-hard [10].

The idea of fuzzy programming in the present context then is that instead of solely working on input representations, one might just as well work on programming the mapping function with the training set conditioning it instead of training it as is the case using neural networks.

The reason that fuzzy mapping may be superior to the use of neural networks is that the more domain-specific knowledge that one can bring to bear on the solution of a problem, the better. Using neural networks, such knowledge takes the form of conditioning the input representation. Using fuzzy mapping, knowledge can additionally be inserted in the form of constraints on acquired patterns.

Case Association

At the level of digital hardware, associative memories operate by matching a (masked) bit pattern against a memory of candidate patterns and retrieving one or more responders. However, this is a very primitive associative memory. A more complex associative memory requires the use of a linguistic mechanism to describe relational and hierarchical content. For example, consider the associative retrieval of all paintings that look like a Monet, or the associative retrieval of all objects that inherit from more than one superclass (e.g., cars that inherit four-wheel drives from trucks and rack-and-pinion suspension from trains).

Logically associative memories (LAMs) work at the level of the human mind and as such are postulated to be excellent for tutoring purposes. For example, consider the tutorial request, "Please explain indefinite integrals using a graphical analogy." In keeping with the notion that a LAM and the human mind are similar in principle it is proposed that LAMs are ideally suited for the construction of decision aids.

Ideally, the user could describe a class of objects using NL, as clarified through the use of a question-answering capability, to specify a logical association and have the computer search through one or more case bases to retrieve the desired material. Here, query is mapped to object. The object is not customized; although, such may be possible through the insertion of knowledge-based technology. Learning may be said to occur in the mapping of the query to the attributes of the retrieved object(s).

Query languages here may be natural languages, visual languages, or any combination thereof. A key concept is conversational learning. That is, for example, if one were to state, "Retrieve an algorithm for multiplication," two scenarios are possible. First, the system might have knowledge of the context and use this knowledge to disambiguate an algorithm for ordinary multiplication (e.g., the Russian Peasants algorithm) from one for matrix multiplication. Of course, one could also multiply a matrix by a constant. Computer languages are typed and typing facilitates disambiguation (casting aside). Natural languages are not typed and where a context is not developed, one can be acquired by way of interactive query. "Do you wish to multiply matrices or scalars?"

Queries need to be mapped to semantics. This is to be accomplished using randomization. A space of queries is mapped to the same semantics. More specifically, a knowledge base may be grown that transforms NL to its semantics. Such a knowledge base is never closed. Rather, attention must be given to the knowledge acquisition bottleneck. Kaser technology [5] may be applied to the amplification of a knowledge base.

Veristic Calculus

LAMs are supported by a veristic calculus, which is a formalism for computing with words [1] [6]. Unlike the predicate calculus, which is its cousin, the veristic calculus is capable of inductive and analogical reasoning in addition to deductive reasoning. For example, "manual" and "Toyota" might be two keywords used to describe a vehicle. "Not an automatic" and "looks like a Corolla" might be two key phrases used to retrieve the former. Any map can be learned in a linear fashion. The key is to learn in a non-linear fashion (i.e., learn more than told). The kaser has this capability. Applying the kaser technology to the LAM requires the evolution of a declarative object tree. For example, here "looks like a Corolla" is a property of Toyota and "not automatic" deduces manual. Again, such object relations need to be evolved through use. It is also clear that a veristic calculus serves as a better model to capture the semantics of NL than does the predicate calculus. Thus, LAMs can be extended to natural language based query and retrieval. The concept of using natural language as a basis for constructing a blackboard system enters the realm of possibility.

5. CONCLUSIONS

Insight on whether or not the brain relies primarily upon a neural representation, a symbolic one, or neither will enable the construction of ever-more intelligent systems. Moreover, the results of the above discussion may be summarized as follows with implications.

- Domain-specific knowledge is intrinsic to the proper functioning of the brain. We make no attempt to distinguish "hard-wired" from acquired knowledge here.
- Knowledge must be communicable, transformable, verifiable, and modifiable. This implies that the brain must be capable of dynamic representation. Thus, the choice between neat and scruffy methods becomes arbitrary and solvable through adaptive processes.
- The need for domain-specific knowledge subsumes the need for domain-specific languages.
- Knowledge must be capable of undergoing amplification. This capability in turn implies an object-oriented hierarchical representation in accordance with the results of our kaser project [5].
- Commonsense reasoning may be effected by a kaser.
- Strong reasoning may be effected by fuzzy mining.
- Fuzzy mapping may be a non-learning kaser. For example, fuzzy mapping could pre-process visual images; whereas, a kaser could learn to associate a semantics with these images.
- Case association models the human associative memory. While the model allows for concurrency, it is not based on parametric equations as are neural nets. Rather, the model is logically associative and thus capable of reasoning. A mechanics for this reasoning will necessarily await a future paper.

If there is a common theme that emerges from these discussions it is this: The brain is an engine for information compression (i.e., randomization). That compression may be partly syntactic, partly semantic, partly loss-less, and partly 'loss'. Representations must be dynamically chosen to facilitate randomization operations. Sometimes these representations will be neat and sometimes they will be scruffy. The choice is based on domain-specific considerations in conjunction with evolutionary processes.

In conclusion, we can't know the mechanics of the brain for certain – only that in accordance with our central thesis it operates to compress information and thereby decrease the overall entropy of the system. We should follow this model in the construction of large-scale creative systems to achieve an ever-greater artificial intelligence.

6. ACKNOWLEDGMENTS

This paper includes the work of U.S. Government employees performed in the course of their employment and is not subject to copyright. It is approved for public release with an unlimited distribution. The Type I and II kaser are the exclusive work of Stuart H. Rubin. They have been assigned to the U.S. Navy, patents pending. This work was supported in part by the Office of Naval Research, NSERC, and the BC Advanced Systems Institute Fellowship.

7. REFERENCES

[1] S.H. Rubin, "Computing with Words," *IEEE Trans. Syst. Man, Cybern.*, vol. 29, no. 4, pp. 518-524, 1999.

[2] E.A. Feigenbaum and P. McCorduck, *The Fifth Generation*. Reading, MA: Addison-Wesley Publishing Co., 1983.

[3] G.J. Chaitin, "Randomness and Mathematical Proof," *Scientific American*, Vol. 232, 1975, pp. 47-52.

[4] S.H. Rubin and L. Trajkovic, "On the Role of Randomization in Software Engineering," *The Intern. Conf. on Comp. & Indus. Engr. (28th ICC&IE)*, Cocoa Beach, FL, to appear in 2001.

[5] S.H. Rubin, "On Knowledge Amplification by Structured Expert Randomization (Kaser)," *SSC San Diego Biennial Review*, to appear in 2001.

[6] L.A. Zadeh, "From Computing with Numbers to Computing with Words – From Manipulation of Measurements to Manipulation of Perceptions," *IEEE Trans. Ckt. and Systems*, vol. 45, no. 1, pp. 105-119, 1999.

[7] S.H. Rubin, "A Heuristic Logic for Randomization in Fuzzy Mining," *J. Control and Intell. Systems*, vol. 27, no. 1, pp. 26-39, 1999.

[8] S.H. Rubin, "A Fuzzy Approach Towards Inferential Data Mining," *Computers and Industrial Engineering*, vol. 35, nos. 1-2, pp. 267-270, 1998.

[9] R. L. Harvey, Neural Network Principles, NJ: Prentice-Hall Inc., 1994.

[10] J-H. Lin and J.S. Vitter, "Complexity Results on Learning by Neural Nets," *Machine Learning*, vol. 6, no. 3, pp. 211-230, 1991.