Learning Conceptual Chess for Testing Evolutionary Programming versus a Reasoning-Based Soft Expert System: the KASER

S.H. Rubin[1], G. Lee[2], C. Sivaramakrishna[2], W. Pedrycz[3], and S.C. Chen[4]

[1]SPAWAR - Systems Center
53560 Hull Street
San Diego, CA 92152

[2]Dept. of Electrical & Computer Engr
San Diego State Univ.
5500 Campanile Dr.
San Diego, CA 92182

[3]Dept of Electrical & Computer Engr
University of Alberta
Edmonton, Alberta
Canada T6G 2R3

[4]School of Comp & Information Science
Florida Int'l Univ.
11200 SW 8th St
Miami, FL 33199

ABSTRACT

A soft expert system is one that is qualitatively fuzzy. In this paper, we present such a system known as the "Knowledge Amplification by Structural Expert Randomization" system or KASER. This system facilitates reasoning using a domain specific expert and commonsense knowledge. It accomplishes this through object-classed predicates and an associated inference engine. The KASER addresses the high cost associated with the bottleneck of knowledge acquisition. Further, it also enables the entry of a basis of rules and provides for the automatic extension of that basis through domain symmetries.

We will demonstrate the learning features of the KASER by comparing its capabilities with an evolutionary programming system that tries to learn the game of chess. In this paper, we concentrate on the evolutionary chess player and also describe the learning capabilities of the KASER, found through other tests. While this EP system may be able to play chess, the KASER provides knowledge as to why certain moves are employed as it learns the game. This powerful characteristic allows the KASER to learn supra-linearly, rather than through exhaustive searches. Thus, the KASER can be applied for many other scenarios in which learning through knowledge acquisition is employed.

Keywords: intelligent learning systems, knowledge acquisition, soft expert system

1. INTRODUCTION

The theory of randomization was first published by Chaitin and Kolmogorov [1] in 1975. Their work may be seen as a consequence of Gödel's Incompleteness Theorem [2] in that it shows that were it not for essential incompleteness, then a universal knowledge base could, in principle, be constructed, i.e., one needs to employ just a referential search. Lin and Vitter [3] proved that learning must be domain-specific to be tractable. Production rules can be expressed in the form of: situation $\rightarrow$ action. Such rules, once discovered to be in error, are corrected through knowledge acquisition. Conventionally, a new rule must be acquired for each correction. This is, however, linear learning. Learning how to learn is fundamentally dependent on representing the knowledge in the form of a society of experts. Minsky [4] and Rubin [5] independently provided compelling evidence that the representational formalism itself must be included in the definition of domain-specific learning if it is to be scalable.

The KASER is a knowledge amplifier based on the principle of structured expert randomization. This principle refers to the use of basic (fundamental) knowledge in the capture and reduction of a larger, dependent space of knowledge (not excluding self-reference). In the KASER, the user supplies declarative knowledge in the form of a semantic tree using single inheritance. Unlike conventional intelligent systems, KASERs are capable of accelerated learning in symmetric domains [6-7].

Conventional expert systems generate cost curves below the breakeven line. In conventional expert systems, cost increases with scale and the *increase* is never better than linear. In the case of KASER

systems, the cost *decreases* with scale and is always better than linear, unless the domain is asymmetric (random). A perfectly random domain would have no embedded patterns (true random numbers), while a perfectly symmetric domain would be infinitely compressible (free of information content). Clearly, such constructs are strictly artificial. The more symmetric is the operational domain, the less the cost of knowledge acquisition.

As a synopsis of the KASER, define a production rule to be an ordered pair whose first member is a set of antecedent predicates and whose second member is an ordered list of consequent predicates. Predicates can be numbers or words [8-9].

Previously unknown words or phrases can be recursively defined in terms of known ones. For example, the moves for the Queen in chess (unknown) can be defined in terms of the union of the moves for a Bishop and for a Rook. This is a union of property lists. Other set operations may likewise be used (intersection, difference.). The use of fuzzy set operators here (“almost the same as”) pertains again to computing with words [8-[10].

KASER systems can be classified as Type I and Type II, depending on their characteristics. In a Type I KASER, words and phrases are entered through the pull-down menus. The user is not allowed to enter new words or phrases if an equivalent semantics already exists in the menu. In a Type II KASER, distinct syntax may be equated to yield the equivalent normalized semantics. The idea in a Type II KASER is to ameliorate the inconvenience of using a data entry menu with scale. In a Type II KASER, selection lists are replaced with semantic equations from which the list problem is automatically solved.

Thus a KASER system can amplify a knowledge base. It represents an advance in the design of intelligent systems because of its capability for symbolic learning and qualitative fuzziness. In a conventional expert system, the context may cover the candidate rule antecedent, in which case an agenda mechanism is used to decide which matched rule to fire (most-specific match, first to match, chance match.). The KASER system follows the same rule-firing principle – only the pattern-matching algorithm is necessarily more complex and embeds the conventional approach as its degenerate case.

The structure of a KASER is summarized as follows:

1. First, antecedent menus are used to specify a contextual conjunct. This results in contextual normalization being realized in linear time in terms of the number of conjuncts and the depth of search.
2. Secondly, it computes the specific stochastic measure, which is indicative of the degree of predicate instantiation taken in the antecedent tree.
3. Next, the system exits the matching process with success (for a row) or failure based on reaching the primitive levels, a timer-interrupt, a forced interrupt, and/or using the maximum allocated memory.
4. Then, if a sequence of consequent actions has been attached, the sequence is pushed onto a stack in reverse order such that each item on the stack is expanded in a depth-first manner.
5. If a match is not found, then, since an expanded rule antecedent is already present, the context is expanded in a breadth-first manner (if enabled by the level of permitted generalization). The general stochastic measure, a measure of validity.
6. The system exits the matching process with success (for the entire current context for a row) or failure based on reaching the primitive levels, a timer-interrupt, a forced interrupt, and/or using the maximum allocated memory. It is permissible to have a concept appear more than once for reasons of economy of reference, or to minimize the stochastic measures (provide confirming feedback). The stochastic measures also reflect the rapidity with which a concept can be retrieved if not cached.
7. For the knowledge acquisition:

a) New rules are added at the head.

b) If exit occurs with failure, or the user deems a selected consequent (in a sequence of consequents) to be in error, then the user navigates the consequent menus to select an attached consequent sequence, which is appropriate for the currently normalized context.

c) If the user deems that the selected “primitive” consequent needs to be rendered more specific, then a new row is opened in the grammar and the user navigates the consequent menus to select a consequent sequence to attach.

d) A consequent sequence can pose a question, which serves to direct the user to enter a more specific context for the next iteration (conversational learning). Questions usually only add to the context to prevent the possibility of add/delete cycles.

e) The system asks the user to eliminate as many specific terms from the context as possible.

f) The system verifies for the user all the other rule antecedents, in the current row, that would fire, or be fired by the possibly over-generalized context, if matched.

g) A selected consequent number may not have appeared on the trace path so far with respect to the expansion of each consequent element taken individually. Checking here prevents cycle formation.

h) It should never be necessary to delete the LFU consequent in view of reuse, domain specificity, processor speed, and available memory relative to processor speed. If memory space is a premium, then a hierarchy of caches could be used to avoid deletions.

8. A metaphorical explanation subsystem can use the antecedent/consequent trees to provide analogs and generalizations for explanative purposes. The antecedent/consequent paths serve to explain the recommended action in a way similar to the use of the antecedent and consequent menus. The antecedent/consequent menus provide disjunction and "user-help" to explain any level of action on the path. The system inherently implements a fuzzy logic [4, 9-11], based on the use of conjuncts, descriptive phrases, and tree structures. The virtual rule base is exponentially larger than the real one and only limited by the number of levels in the trees as well as by space-time limitations on breadth-first search imposed by the hardware.

9. A consequent element could be a "do-nothing" element if need be. The provision for a sequence of consequents balances the provision for multiple antecedents. The selected consequent(s) need to be as general class objects as can be to maximize the number of levels and thus the potential for reuse at each level.

10. Unlike the case for conventional expert systems, a KASER cannot be used to backtrack consequents (goal states) to find multiple candidate antecedents (start states). The problem is that the pre-image of a typical goal state cannot be effectively constrained (other than for the case where the general and specific stochastic are both zero) in as much as the system is qualitatively fuzzy. The answer is to use fuzzy programming in the forward-chained solution [12]. This approach allows the user to enter the constraint knowledge that he/she has into the search.

The search may be manually terminated by a user interrupt at any time. The search is not to be automatically terminated subsequent to the production of some limit of contexts because to do so would leave a necessarily skewed distribution of contexts, giving the user a false sense of completeness. A terminated search means that the user either needs to use a faster computer, or more likely to just narrow down the search space further and resubmit the query.

II. AN EVOLUTIONARY PROGRAMMING CHESS SYSTEM

In the project presented in this paper, we would like to compare a prototype KASER system with an evolutionary programming (EP) chess system. Again the intent is not to build a better chess machine; rather the intent is to see how learning can be achieved and thus validate the hypothesis that the KASER is a better learning system.

Chess variables include the number of moves until the player loses, the number of wins for each player, the number of erroneous rules induced by the player, or the number of questions, if applicable, by each player plotted as a function of the number of games played. Here, we are looking to see if the KASER can learn to induce suggestions for a qualitative assessment of its performance versus an evolutionary learning algorithm. For the KASER will play chess using cognitive representations of knowledge. In chess, this means that there will be little, if any, search performed. Here, search control knowledge is compressed into heuristics and we create a vector of chess features. These can be developed by a reasonably skilled chess-player using commonsense reasoning, though arduous.

For the evolutionary chess system, we have programmed a feature-based evolutionary algorithm as follows. Pseudo features and procedures or actions have been constructed and described in laymen's terms.

Some sample *pseudo*-features (i.e., concepts used to create features) are:

1. Nearness of the King to the center of the board.
2. Difference in value between the piece that the player could take and the piece that the opponent could take (using standard chess valuations).
3. Number of squares that the player's piece can cover.
4. Number of squares that the opponent's piece can cover.

Note that features may be symmetrically designed as in #3 and #4 above to the maximal extent possible to best capture how humans cognate them.

An *actual* feature may be: *"My Queen covers more squares than does my opponents"*. Thus, we would exclude for example a feature such as, "Does the proposed move give me an advantage n-ply moves ahead?" To run the comparison, we compress the search into static heuristic features. An upper bound on the number of features is 50 and the number of procedures is 20. We select a relatively large number of features to be fair to testing the capabilities of the KASER which will be done as a future activity.

Procedures are used to define legal moves, for example:

1. Take the piece with the highest value, if any. Note that a rule will be expunged and the next rule as determined by the inference engine considered, if any, if its procedure does not result in a legal move. A rule is not considered "fired" if its procedure is not legal and executable for any reason.
2. Take the piece closest to the center of the board, if possible (we use a Manhattan metric to realize this).
3. Take the piece having the highest differential value, if possible (i.e., a Pawn should preferentially take the Queen).
4. Advance the King's Knight's Pawn by two, if permissible (observe that it is not a legal move if the opponents Pawn blocks advancement).

Now, legal boards are generated by the chess software. The level of chess play will be determined by the interaction of the rules in the knowledgebase. Rules are expunged whenever they would otherwise make an illegal move. This need not cause the game to restart as the inference engine can automatically select another rule in this event (i.e., unless the base is empty of course).

A sample actual rule may be:

R1: IF my Queen covers less squares than does my opponents AND opponents Queen is closer to the center of the board than mine THEN capture opponents Queen using same or lesser-valued piece, if possible.

In the EP study, to generate the best EP opponent, the winning base is used to play against a random mutation of itself – defined by an arbitrary number (and loci) of rule deletions, replacements, and additions in that order. To avoid the mesa (hill-climbing) phenomenon, and in keeping with evolutionary programming (EP) principles, several rules are evolved simultaneously to get a population for mutation. This process is iterated until a better rule base cannot be evolved after some predefined number of trials.

After completing the evolutionary programming architecture in the chess evolver and after creating a non-redundant chess features and procedures, the next step was to initiate the lists of movements on the chess board which is divided into two phases.

1. Part A emphasizes the capturing the movements of the chess pieces by calling a event handler and thus obtaining the individual positions of each piece on the board. Thus a link from the mouse click to the electronic board provides movements on the board.
2. Part B checks for the legality of each move done by the mouse handler on the chess board. Hence after each move, the EP, which has already been programmed for generating features and procedures defined in the knowledgebase, is triggered to check for legal steps.

The knowledgebase, which was created using a random generator to create the distinct rule bases by a uniform chance, are coded in Java to create a unique database, i.e., each trigger/firing of a feature from the knowledgebase is interfaced with the Chess Evolver to play the game of chess. The game controller can thus be created to control the chess pieces for each game. Thus, after each completed game, results of the previously designed Evolutionary Program were used to obtain an improved knowledgebase. This

repetitive process will be carried out for n iterations (e.g., number of games) to create the best legal rules which are used to form the final knowledgebase. This knowledgebase, generated through EP, will be the reference opponent for the T2K in benchmarking the learning characteristics of the KASER.

III. RESULTS

The EP chess algorithm was tested by initially constructing two EP chess players, using a random generation of sets of rules. The winner of the first set of games was then evolved to construct a new player, to play against the "best" current EP chess player. The winner of the second set of games was evolved to form a new player to play against the "best" current EP player, and so on.

In this way through each evolution after a set of games, one can formulate a trend of how the EP chess algorithm is learning. Note that the reason for this approach for testing learning provides the same benchmark for the KASER testing as well as for the testing between the EP player AND the KASER.

Figure 1 shows the learning trend for the EP algorithm. For this particular experiment, 18 features, 30 procedures, and 35 rules were generated. The number of features and procedures selected for the rules were randomly assigned and the number of rules kept for the evolved chess player was randomly selected. Notice that learning is essentially linear in that, although the next EP chess player has knowledge based upon previous wins and losses, the learning is not accelerated and reaches a saturation learning point.

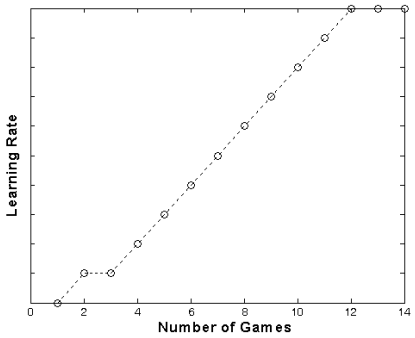

Figure 1: Learning Trends using Evolutionary Programming

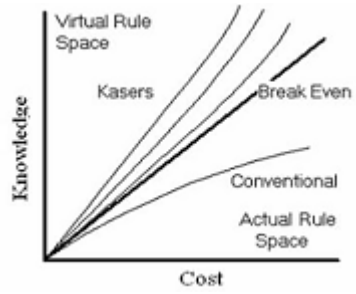

Figure 2: Expected Performance of KASERs versus Conventional Learning Approaches

The KASER will be supplied with the *same* initial iterate rule base for a more meaningful comparison. The EP chess provides a rule-base dump to compare with the KASER learning. Notice that unlike so-called, "chess machines", our EP chess is capable of providing symbolic explanations for its moves by replaying the fired rules in sequence – using what is termed, the *explanation subsystem*. Then, in as much at the KASER learns from and amplifies upon user feedback, the following adjustments will be made. Whenever the user corrects a KASER rule, add no more than (i.e., an upper bound of) one game to its statistic for purposes of comparison with the random game evolver. This corrects for user-supplied knowledge, since each randomly evolved rule base is more or less better than its predecessor.

It is expected that, given the learning trends of the EP player and the learning tends of the KASER on other test scenarios, the KASER will outperform the EP algorithm (Figure 2). This is the next step in the research effort discussed here.

IV. CONCLUDING REMARKS AND FUTURE RESEARCH

The objectives of this paper are to summarize the learning features of an evolutionary programming chess system, evolved through randomization to accelerate the learning process, and to suggest how these features will compare to a KASER system that is currently being built – also to be evolved through

randomization to accelerate the learning process. It is expected that the KASER will outperform the EP program using several evaluation metrics. The KASER has exhibited domain transference and supra-linear learning, which bear proportion to the inherent degree of domain symmetry. It introduces the use of single inheritance declarative object trees in expert systems. Such trees facilitate the capture of object-oriented semantic relations among rule predicates and serve the processes of metaphorical explanation as a consequence. Such learning features show that the KASER is superior to classical learning mechanisms through transformational knowledge and randomization.

This will be validated by way of two experimental studies. In the first study, the KASER will learn chess by playing against a commercially available chess game. Absolutely no knowledge of the game is provided except that conveyed through the use of structured descriptions of the play and of course the definitions of legal moves.

In the second study, the KASER will learn chess by playing against an evolutionary programming system, developed through many games played against another EP system. The best EP system will be used as the opponent to the KASER system.

It is anticipated that these activities will illustrate that the KASER or Knowledge Amplification by Structural Expert Randomization learns supra-linearly, thus taking machine learning to new heights.

V. REFERENCES

[1] G.J. Chaitin, "Randomness and Mathematical Proof," *Scientific American*, vol. 232, no. 5, pp. 47-52, 1975.

[2] V.A. Uspenskii, Gödel's Incompleteness Theorem, Translated from Russian. Moscow: Ves Mir Publishers, 1987.

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

[4] M. Minsky, *The Society of Minds*, Simon and Schuster, New York, 1987.

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

[6] S.H. Rubin, S. and G. Lee, "On the Use of Randomization for System of Systems (SoS) Design of Intelligent Machines**"**, *Proc. of the World Automation Congress*, ISSCI, Budapest, 2006.

[7] S.H. Rubin and G. Lee, "Learning Using an Information Fusion Approach", *Proc. of the ISCA Int'l Conference on Intelligent and Adaptive Systems*, Nice, 2004.

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

[9] S.H. Rubin, R.J. Rush, Jr., J. Boerke, and L.J. Trajkovic, "On the Role of Informed Search in Veristic Computing," *Proc. of the 2001 IEEE Int. Conf. Syst., Man, Cybern.*, pp. 2301-2308, 2001.

[10] S.H. Rubin, R.J. Rush Jr., J. Murthy, M.H. Smith, and L.J. Trajkovic, "KASER: A Qualitatively Fuzzy Object-Oriented Inference Engine," *Proc. of the North American Fuzzy Information Proc. Soc.*, pp. 354-359, 2002.

[11] S.H. Rubin, G. Lee, W. Pedrycz and S.C. Chen, "Modeling Human Cognition Using a Transformational Knowledge Architecture", *Proc. of the IEEE Intl. Conference on System of Systems Engineering (SoSE)*, Monterey, CA, 2008.

[12] S.H. Rubin, "A Fuzzy Approach Towards Inferential Data Mining," *Computers and Industrial Engineering*, vol. 35, no. 1-2, pp. 267-270, 1998.