A Sparse Distributed Memory Capable of Handling Small Cues, SDMSCue

Ashraf Anwar and Stan Franklin
College of Computing and Information Technology, Arab Academy for Science, Technology and Maritime Transport, Alexandria, Egypt, dr.a.anwar@ccit.aast.edu; Computer Science Department, University of Memphis, Memphis, TN 38152, franklin@memphis.edu

Abstract: In this work, we present Sparse Distributed Memory for Small Cues (SDMSCue), a new variant of Sparse Distributed Memory (SDM) that is capable of handling small cues. SDM is a content-addressable memory technique that relies on similar memory items tending to be clustered together in the same region or subspace of the semantic space. SDM has been used before as associative memory or control structure for software agents. In this context, small cues refer to input cues that are presented to SDM for reading associations; but have many missing parts or fields from them. The original SDM failed to handle such a problem. Hence, our work with SDMSCue comes to overcome this pitfall. The main idea in our work; is the projection of the semantic space on a smaller subspace; that is selected based on the input cue pattern, to allow for read/write using an input cue that is missing a large portion. The test results show that SDMSCue is capable of recovering and recalling information from memory using an arbitrary small part of that information; when the original SDM would fail. SDMSCue is augmented with the use of genetic algorithms for memory allocation and initialization. We think that the introduction of SDMSCue opens the door to more research areas and practical uses for SDM in general.

Keywords: Artificial Intelligence; Cognition; Memory; SDM; SDMSCue; Software Agents.

1. INTRODUCTION

Sparse Distributed Memory (SDM) is a content addressable memory developed by Kanerva. SDM was proposed to be a tool and model of human associative memory (Kanerva, 1988a; Kanerva & Raugh, 1987).

SDM has proven successful in modeling associative memories (Anwar, 1997; Anwar & Franklin, 2003; Rao & Fuentes, 1996; 1998; Scott, Fuller, & O'Brien, 1993). Associative memory is typically needed for intelligent and cognitive autonomous agents (Glenberg, 1997; Kosslyn, 1992). In particular, both cognitive software agents (Franklin, 1997; 2001) and "conscious" software agents (Franklin & Graesser, 1999) need such a memory. One of the "conscious" software agents, which we did work with, IDA: Intelligent Distribution Agent, uses SDMSCue (Franklin, Kelemen, & McCauley, 1998). The use of SDMSCue in IDA is to learn and keep associations between various pieces of information pertaining to the task of personnel distribution.

2. THE MOTIVE FOR SDMSCue

In many cases the need for associative memory to be able to handle and retrieve associations based on arbitrary small cues is crucial. For example; in IDA, we are often faced with a situation in which we need to retrieve associations based on very small pieces of information like part of email address, part of name, or social security number. Humans have no problem retrieving associations based on arbitrary small cues. While the original SDM modeled many aspects of human memory very successfully, it failed miserably in dealing with the issue of retrieving associations for short-length or small cues. Without such a capability, we are missing a key human-like feature in associative memory models that are based on SDM. Hence, the role of SDMSCue comes to the scene.

SDMSCue uses an elegant space projection mechanism to *enlarge* the short-length input cue successively until it is large enough for a read/write from/to the entire full-length SDM semantic space. The enlargement process uses successively increasing subspaces for reads/writes. To be noted is that both read and write operations in SDM involve the selection of an access circle to read from, or to write to. The selection is typically based on similarity between the input read/write cue, and the hard locations addresses within the access circle.

3. SPARSE DISTRIBUTED MEMORY

Sparse Distributed Memory, SDM, is the work of Pentti Kanerva (1988a, 1988b). It gets its name from the sparse allocation of storage locations in a vast binary address space and from the distributed nature of information storage and retrieval. A typical SDM has a vast binary space of possible memory locations in a 2^n semantic space where *n* is both the full word length and the dimension of the address space. For any practical application, only a very small portion of this 2^n semantic space can actually exist. For more details and discussions, see (Anwar, 1997). Also see (Franklin, 1995) for a brief overview, and (Willshaw, 1990) for a useful commentary. Many evaluations, extensions, and enhancements have been suggested for SDM (Evans & Surkan, 1991; Karlsson, 1995; Kristoferson, 1995a; 1995b; Rogers, 1988a; 1988b; Ryan & Andreae, 1995). A more efficient initialization technique for SDM using Genetic Algorithms was also suggested (Anwar, Dasgupta, & Franklin, 1999).

There are two main types of associative memory, auto-associative and hetero-associative. In auto-associative memory, a memory item, typically with noise and/or missing parts, is used to retrieve itself. In hetero-associative memory, memory items are stored in sequences where one item leads to the next item in the sequence. The auto-associative version of SDM is truly an associative memory technique where the contents and the addresses belong to the same space and are used alternatively.

A Boolean space is the set of all Boolean vectors (points) of some fixed length, *n*, called the dimension of the space. The Boolean space of dimension *n* contains 2^n Boolean vectors, each of length *n*. The number of points increases exponentially as the dimension increases. Boolean geometry uses a metric called *Hamming Distance,* where the distance between two points is the number of coordinates at which they differ. Thus $d((1,0,0,1,0), (1,0,1,1,1)) = 2$. The distance between two points will measure the similarity between two memory items, closer points being more similar. We may think of these Boolean vectors as feature vectors, where each feature can be only on, 1, or off, 0. Two such feature vectors are closer together if more of their features are the same.

The word length, which is also the dimension of the space, determines how rich in features each word and the overall semantic space are. Features are represented by one or more bits in a Boolean vector or binary string of length *n*. Groups of features are concatenated to form a word, which becomes a candidate for writing/reading into/from SDM. Another important factor in SDM design; is how many real memory locations are implemented. These are called *hard locations*. When writing, a copy of the object binary string is placed in all -close enough- hard locations. When reading, a subject cue would reach all close enough hard locations and get some sort of aggregate or average word from them. Reading is not always successful (Anwar, 1997; Kanerva, 1988a). Depending on the cue and the previously written information, among other factors, convergence or divergence during an iterative reading operation may occur. If convergence occurs, the pooled word will be the closest match of the input reading cue, possibly with some sort of abstraction. On the other hand, when divergence occurs, there is no relation, in general, between the input cue and what is retrieved from SDM.

SDM is a content addressable memory that, in many ways, is ideal for use as a long-term associative memory. "Content addressable" means that items in memory can be retrieved by using all or part of their contents as an address or a cue; rather than having to know their actual addresses in memory as in traditional computer memory.

Envisioning the semantic space as a sphere around an arbitrary point, and for n sufficiently large; most of the address space lies midway in the sphere from the point at the center of the sphere (Kanerva, 1988a). In other words, almost all the space is far away from any given point.

A Boolean space implementation is typically sparsely populated for sufficiently large n; an important property for the construction of SDM, and the source of part of its name. For n=1000, one cannot hope to actually implement such a vast 2^n memory in its entirety. On the other hand, considering humans and feature vectors, a thousand features wouldn't deal with just human visual input until a high level of abstraction has been reached. A dimension of 1000 may not be all that much for real-life cognition; it may, for some purposes, be unrealistically small. Kanerva proposes to deal with this vast address space by choosing a uniform random sample, of size 2^{20} for n = 1000, of hard locations. An even better way to distribute the set of hard locations over the vast semantic space using Genetic Algorithms was suggested (Anwar, 1999). With 2^{20} hard locations out of 2^{1000} possible semantic locations, the ratio is 2^{-980}, *truly sparse*.

SDM is distributed in that many hard locations participate in storing and retrieving (write/read) each word, and one hard location can be involved in the storage and retrieval of many words. This is very different from the one-word-per-location type of memory to which we are accustomed. For n = 1000, each hard location, stores data in 1000 counters, each with range from –K to K, where K is an implementation-dependent factor that determines memory capacity. We now have about a million hard locations, each with a thousand counters, totaling a billion counters in all. For K = 40, numbers in the range -40 to 40 will take most of a byte to store. Thus we are talking about a billion bytes, a gigabyte, of memory.

Counters are updated according to the words written. Writing 1 to one counter increments it; writing 0 decrements it. To write a word at a given hard location x, write each coordinate of the word into the corresponding counter in x; either incrementing it or decrementing it.

At any arbitrary location, the sphere centered at that location, is called the access sphere or access circle of that location. For n = 1000, and 2^{20} hard locations, an access sphere typically contains about a thousand hard locations, with the closest usually some 424 bits away and the median distance from the center point to hard locations in the access sphere about 448. Any hard location in the access sphere is said to be *accessible* from the center point address or word. With this machinery in hand, any location -hard or not- can be written to in a distributive manner. To write a word to a location, simply write the word to each of the roughly one thousand hard locations within the location access sphere, i.e. accessible from the location as a center point.

We read from an arbitrary point or address in the semantic space. This read includes reading from all hard locations within the access sphere from that point. To read from a single hard location, x, we compute the bit vector read at x, by assigning its i^{th} bit the value 1 or 0 according as the i^{th} counter at x is positive or negative. Thus, each bit results from a majority rule decision of all the data that have been written at x. The read word is an archetype of the words that have been written to x, but may not be any one of them. For any location or address, the bit vector read is formed by pooling the data read from each hard location accessible from that location or address. Each bit of the read word results from a majority rule decision over the pooled data. The voting is influenced by n stored copies of the word, and about $(10\ n)$ other stored data items. Since the intersection of two access spheres is typically quite small, the other data items influence a given coordinate only in small groups of ones or zeros, which tend to compensate for each other, i.e. white noise. The n copies of the original word drown out this slight noise.

The entire stored word is not needed to recover itself. Iterative reading allows recovery when reading from a noisy version of what has been stored. There are conditions, involving how much of the stored word is available for the read operation; under which this is true, (Kanerva, 1988a). Reading with a cue that has never been written to SDM before gives, if convergent, the closest match stored in SDM to that input cue, with some sort of abstraction if close items have been written to the memory. SDM works well for reconstructing individual memories (Hely, 1994).

4. SDM FOR SMALL CUES (SDMSCue)

4.1 Approach

Using a variant of SDM capable of handling small cues, we are able to overcome the main shortage in Kanerva's model (Kanerva, 1988a; 1992). One of the main problems with Kanerva's SDM is that the input cue has to be of sufficient length to be able to retrieve a match. The reason is that the *entire* input cue is considered and the hamming distance between its *entire* binary string representation and various hard locations in the access sphere or access circle is considered when reading or writing. So if we have a small cue, the missing large part is almost guaranteed to sink the known small cue in terms of hamming distance, thus being indifferent to all words or hard locations in the SDM memory.

In many cases, we are faced with a very distinguished and unique memory cue that is considerably small in size than the typical 65-80% requirement in Kanerva's work. We -humans- are able to retrieve relevant information associated with such a small cue efficiently. For SDM to be able to function similarly, we need a variant of SDM that is capable of handling small cues. When faced with retrieval based on a substantially smaller cue like part of a name, part of an email address, or SSN, this calls for the use of SDMSCue.

The goal is to retrieve appropriate corresponding word matching such a small cue. Using subspaces with increasing sizes in a progressive way, we are able to read and retrieve the whole original corresponding memory item using only a small portion of the cue with arbitrary small length. However, the number of levels needed for read/write depends upon the original size of the small cue. The smaller the original input cue is; the more levels of read/write needed. Time complexity of the operation is linearly proportional to the number of levels needed.

4.2 Design of SDMSCue

The idea is to project the original SDM semantic space onto a smaller subspace corresponding to the small cue. In such a projection, only memory locations with ***matching*** content to the small input cue contribute to Read/Write operations.

By projecting the space onto a smaller subspace, we are able to use the smaller subspace for much higher recall rate for a considerably small cue. The gain occurs mainly because of constraining and limiting only hard locations that match the small cue part and allowing only them to contribute to the subspace for read/write operations.

The result obtained from a read/write operation at one stage; is used to access a larger subspace including the input cue along with associations retrieved that typically range from 25% to 35% of the former input cue length. Such associations are retrieved from the contents of the hard locations that were selected, and contributed to the former subspace read/write.

By repeating the above process for increasingly larger subspaces and levels of projection, we eventually get to access the entire semantic space for read/write.

To be noted is that during write operations, actual writing to hard locations occurs only at the final level when writing to the entire space. All preceding access takes place for association retrieval only. So both read and write operations are the same (reading and retrieving associations) until the final level when we access the entire space. In both cases association buildup takes place to enlarge the small input cue gradually until the length obtained is large enough to read from the entire semantic space. In the last level or phase, if it is a read operation, we simply retrieve associations and obtain the matching entire word. If it is a write operation, the enlarged input cue is written to all hard locations within the access circle of the last phase, which is part of the entire semantic space.

4.3 How SDMSCue Works

Using SDMSCue, we can manage to access (read/write) with small cues. The process goes in phases in reading or writing operations. When accessing, in the first phase, we read from a small sub-space that corresponds to the input small cue plus extra association information. This read –if convergent- yields a longer word due to the association of information. This resulting word is then used as the input to the second phase. In the second phase, a similar process takes place reading from a larger subspace using the output result from the first phase as input. This process continues until the subspace being read from is the entire original semantic space.

For example, as shown in *Table 1*, we start by reading with a small cue of length 17% of the whole memory word size, using a 0.35 ratio for associations. Then the reading operation yields a larger word, due to adding associations, of length 23% of the whole memory word.

Table 1: Multi-Level Reading operations from SDMSCue, and their corresponding Length Percentage, ***m****. In Level 1, with a small cue of length 17% of the whole memory word, projection of the space and reading yields bigger subspace of 23%. In a similar way successive projection increases the subspace till it reaches the whole space, i.e. 100% of the memory word.*

Level #	Input Word Length	Output Word Length	Output Word
1	17	23	|----------------|
2	23	31	|-----------------------|
3	31	42	|------------------------------|
4	42	57	|---|
5	57	77	|--|
6	77	100	|---|

Then using the 23% retrieved and formed word as input to the second phase and adding 0.35 associations to it, a 31% word is obtained. This process continues until in the final level (6th level in the example), a 77%

retrieved and formed word is used to access (read from or write to) the entire original semantic space.

To be noted is that the time complexity of a Read/Write operation is ***linearly*** proportional to the number of levels involved in a Read/Write. However, the overall effect is quite minor compared to the gain of the approach. Assume an original cue length of m% of the whole memory word size; where **m** ranges from 0 to 100. When reading/writing from SDMSCue, some associations are retrieved for the small cue at each level resulting in a length gain. Let i be the percentage of the length gain at each level. Adding the gain in length, i, to the next input cue in each successive read/write level, the maximum number of read/write levels N is given by:

$100 \leq m * (1 + i)^N$; Last word length needs to be 100%, i.e. last read needs to occur from the entire semantic space

$$\Rightarrow N = \lceil \log(100/m)/\log(1+i) \rceil$$
$$\Rightarrow N = \lceil (2-\log m)/\log(1+i) \rceil$$

For $m = 17$ (17% original small cue length), $i = 0.35$ (35%), as in *Table 1*,
$N = \lceil (2 - \log 17) / \log 1.35 \rceil = \lceil 5.9 \rceil = 6$ Levels.

For $m = 10$ (10% original small cue length), $i = 0.3$ (30%),
$N = \lceil (2 - \log 10) / \log 1.3 \rceil = \lceil 8.78 \rceil = 9$ Levels.

For $m = 1$ (1% original small cue length), $i = 0.3$ (30%),
$N = \lceil (2 - \log 1) / \log 1.3 \rceil = \lceil 17.55 \rceil = 18$ Levels.

We define the term SDMSCue **Latency Factor** to be the average number of levels needed for read/write for a certain word distribution to be written to or read from SDMSCue. Such a factor is both semantic space dependent, and distribution dependent.

SDMSCue makes use of GA for more efficient space initialization and hard locations allocation (Anwar, 1999; Anwar, Dasgupta, & Franklin, 1999). The uniformity of the semantic space is –in general- favorable to better recall rates for SDMSCue as well as SDM.

4.4 SDMSCue Convergence and Divergence

We need to develop a notion for overall convergence and divergence in case of Read/Write from SDMSCue. For overall convergence to occur, all phases or levels of Read/Write must converge. Overall divergence occurs if at any phase or level, the Read/Write diverges. In other words, convergence is the Boolean "AND" operation of the convergence in all levels.

$$\text{Convergence}\,|_{\text{SDMSCue}} = \text{AND}_{\text{For All } i}(\text{convergence}\,|_{\text{at level } i})$$

$$\text{Divergence}\,|_{\text{SDMSCue}} = \text{NOT}\,(\text{Convergence}\,|_{\text{SDMSCue}})$$

4.5 Implementation

The following is a short note about the current implementation of SDM with small cues (SDMSCue). Java Visual Symantec Caf Professional Edition was used for testing and implementation of the code for SDMSCue in Windows XP environment. The hardware was a Pentium 2.4 GHz with 1GB RAM. The results obtained are based on recall performance and memory trace used for comparison tests of SDMSCue vs. original SDM. Runs were performed repetitively 100 times on average for each case.

5. RESULTS AND STATISTICS

The following comparison between SDMSCue and regular SDM was done using the same memory parameters, and memory trace (Loftus & Loftus, 1976). Memory performance in terms of various operational parameters was considered for SDMSCue vs. original SDM. The various memory parameters: Memory Volume, Cue Volume, Similarity, and Noise were considered.

Memory Volume is the average number of features in the memory trace. In other words, it is the average number of 1's in a memory word. It measures the richness of the memory trace. Memory volume is a vital parameter in the distinction of the memory trace. It signifies the distribution of various memory words over the semantic space.

Retrieval Volume is the same as Memory volume but for a single input cue or input word to memory. It has almost the same effect on retrieval as memory volume.

Similarity is a measure of how similar, in average, are the words written to memory. The more similar the words written to memory are, the more clustered contiguously they are, and the harder it is to retrieve them. The hamming distance is the measure of similarity in SDM as well as SDMSCue. The less the hamming distance between two memory words, the more similar the memory words are. However, there is a difference between the similarity of hard locations and the similarity of written memory words. Using genetic algorithms (Anwar, 1999) a uniform distribution of the hard locations in SDM can be obtained.

Noise determines the number of noise bits, on average, in a memory word. It reflects directly on the reliability of retrieval of stored memory words.

Table 2 shows the distribution of the percentage of input cues in memory trace, used for the test, with respect to cue length, measured as percentage of the whole length. For example, according to *Table* 2, 35% of the cues in memory trace do not exceed 20% in length (small cues), while 25% of the cues in memory trace have length greater than 20% but less than or equal to 40% (low medium cues). Also only 10% of the input cues have length greater than 70% (longest cues). The second column gives the number of levels needed for read/write operation using the formula devised in section 4.3. As shown in *Table 2*, for the chosen distribution, the overall average cue length is 36%, and the average number of levels for read/write is 5. So, for the distribution at hand, SDMSCue has a latency factor of 5.

Table 2: The distribution of input cues in memory trace with respect to cue length, and their corresponding number of Read/Write levels.

Cue Minimum-Maximum Length as percentage of the Whole Word	Average Number of Read/Write Levels Needed	Percentage of Cues in Memory Trace
Less than 20%	8	35%
20%-40%	5	25%
40%-50%	3	10%
50%-60%	2	10%
60%-70%	2	10%
70%-100%	1	10%
Average Length 36%	**Latency Factor** = Average Number of Levels = **5**	

To be noted is that this distribution was chosen to illustrate the advantage of using SDMSCue when considerable percentage of the input cues is short in length, i.e. missing too many parts. By no means is this the only distribution that can illustrate the idea, but just the one we settled upon after some trials to illustrate the benefit of using SDMSCue when considerable number of the input cues is short in length. However, varying the distribution will definitely change the gain achieved from using SDMSCue over SDM.

Table 3 shows a comparison between the recall in SDM vs. SDMSCue. Various combinations of the memory trace parameters were considered. Each was varied on a Low/High scale.

Table 3: Comparison between GA initialized SDM, and GA initialized SDMSCue. This is a comparison between the performance of SDM with and without small Cues Capability, and the gain in memory recall resulting from the use of SDMSCue. Memory Volume and Retrieval Volume (H=60%, L=10%). Similarity between Memory Items (H=70%, L=30%). Noise (H=30%, L=10%).

#	Memory Volume	Retrieval Volume	Similarity	Noise	SDM Hit % in Recall	SDMSCue Hit % in Recall	Recall Gain %	Decrease in Miss in Recall Gain %
1	L	L	L	L	6	44	633	40
2	L	L	L	H	5	40	700	37
3	L	L	H	L	7	39	457	34
4	L	L	H	H	5	34	580	31
5	L	H	L	L	9	56	522	52
6	L	H	L	H	8	50	525	46
7	L	H	H	L	11	49	345	43
8	L	H	H	H	5	42	740	39
9	H	L	L	L	74	98	32	92
10	H	L	L	H	66	93	41	79
11	H	L	H	L	61	93	52	82
12	H	L	H	H	54	89	65	76
13	H	H	L	L	73	99	36	96
14	H	H	L	H	56	95	70	89
15	H	H	H	L	60	96	60	90
16	H	H	H	H	50	92	84	84

The gain achieved from using SDMSCue is illustrated in the last two columns. The first gain, Recall Gain, measures the improvement in successful recall or Hit in SDMSCue over original SDM. The second gain, Decrease in Miss in Recall, measures the decrease in Miss in SDMSCue over the original SDM. This gain measures the improvement in SDMSCue over SDM in terms of decrease in the percentage of unsuccessful recall or memory Miss. To be noted is that a T-Test of statistical dependence (Kanji, 1999; Vogt, 1998) shows statistical significance between the recall of SDMSCue and that of SDM.

In *Table* 3, Row 4, which represents poor recall conditions (Low Memory Volume, Low Retrieval Volume, High Similarity, and High Noise), shows large improvement in successful recall in SDMSCue over original SDM. Row 13, on the other hand, represents near optimal recall conditions (High Memory Volume, High Retrieval Volume, Low Similarity, and Low Noise). Row 13 corresponds to 36% recall gain.

For decrease in miss in recall gain, recall conditions play similar role. Row 4, which represents poor recall conditions, shows 31% gain. Row 13, which represents near optimal recall conditions, shows 96% gain.

In general, results in *Table 3* show that the higher the volume of the memory and/or the retrieval volume, the better the recall for both memories, SDMSCue and SDM. However, when the memory volume is quite low, the recall gets really affected. The degradation in performance is more graceful in SDMSCue than it is in SDM. This has to do with the fact that SDMSCue projects the space on the part of the cue that exists, thus greatly moderating the typical negative effect of low memory volume.

With respect to similarity, the more distinct the memory words are, i.e. the less similarity, the better the recall in general. This makes absolute sense since SDM in general; and accordingly SDMSCue as well, uses hamming distance as a measure of inclusion of memory words in access sphere or access circle for read/write. The more similar memory words are, the closer they become in terms of hamming distance. Hence, they tend to get clustered together and cause crowding effect in the semantic space, where they may sink each other in certain regions of the semantic space.

To be noted, though, is that such clustering effect in the semantic space resulting from highly similar items being written to SDM or SDMSCue is somewhat similar but not the same as clustering of hard locations when a poor initialization technique is used for hard locations assignment. The later is more crucial to the functioning of SDM or SDMSCue, and should be application independent, whereas the former depends on the memory trace fed to SDM or SDMSCue, and hence is application dependent by nature. While there is a straightforward way to guarantee uniformity of hard locations assignment in the semantic space of SDM or SDMSCue (Anwar, Dasgupta, & Franklin, 1999), there is no immediate direct solution to overcome clustering in SDM or SDMSCue due to similarity of words or memory items written to it.

As expected, for the effect of noise on recall; the lower the noise, the better the recall in general. Noise can however be sunk to a degree as a result of the distributed nature of read/write, as well as the abstraction achieved from using SDM and SDMSCue.

Figure 1 shown below contrasts the performance of SDMSCue vs. SDM in terms of hit rate in recall. In this comparison, the 16 different memory configurations in *Table 3* were used, e.g. configuration 9 is HLLL which stands for High Memory Volume, Low Retrieval Volume, Low Similarity, and Low Noise. For each configuration, the same memory trace was applied to both SDM and SDMSCue. Then recall was tested with the same set of patterns with lengths distributed according to *Table 2*. Recall hit rate for each memory parameters configuration (recall conditions combination) was calculated and is shown as percentage over the vertical axis in *Figure 1*.

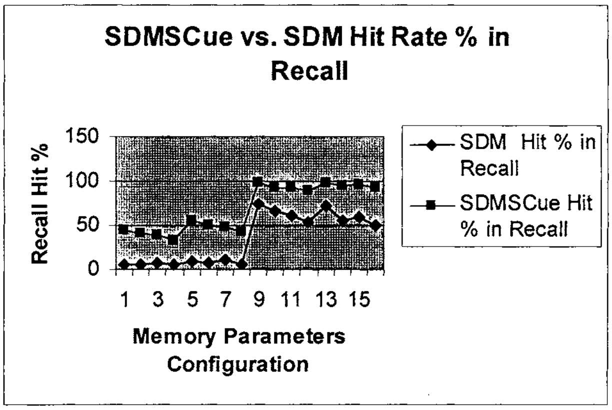

Figure 1: SDMSCue vs. SDM Hit Rate % in Recall. The top line shows the recall hit rate in SDMSCue. The bottom line shows the recall hit rate in SDM. The X-axis is for the 16 different memory parameters configurations from Table 3. The Y-axis is for the percentage of Recall Hit Rate.

It is clear from *Figure 1* that SDMSCue consistently outperforms SDM in terms of recall under all conditions. This comes at no surprise since SDMSCue uses SDM functionality in addition to the elegant space-projection to filter out non-relevant memory locations. SDMSCue also uses a far more superior GA approach for uniform space initialization and allocation of hard locations (Anwar, 1999).

Figure 2 shows the gain of SDMSCue over original SDM using the results and figures from *Figure 1*. The bottom area is the recall gain in SDMSCue over SDM. The gain percentage of SDMSCue over SDM is defined by: 100*(SDMSCue Hit% – SDM Hit%) / (SDM Hit%). For example, for memory parameter configuration 1, the gain is 100 * (44 - 6) / 6 = 633%.

The upper area in *Figure 2* is the improvement in recall measured as the decrease in miss rate in recall. This is defined by: 100*(SDM Miss% – SDMSCue Miss%) / (SDM Miss%). For example, for memory parameter configuration 1, the improvement as decrease in miss rate is 100*(94 – 56) / 94 = 40%. For memory parameter configuration 13, the improvement as decrease in miss rate is 100*(27 – 1) / 27 = 96%.

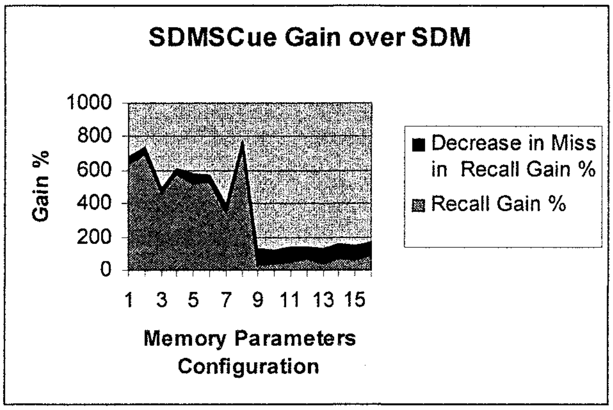

Figure 2: SDMSCue Gain over SDM. The results were obtained using Figure 1. The bottom area shows the gain in recall. The upper area shows the improvement in recall measured as the decrease in miss rate in recall.

6. CONCLUSIONS

SDMSCue is superior to traditional SDM in its capability of handling small cues that original SDM was not able to. One of the major difficulties encountered in using original SDM as an associative memory, is its inability to recover associations based upon relatively small cues; whereas we humans do. For a typical SDM to converge, a sufficiently large portion of a previously written word must be presented to the memory as an address.

The SDMSCue enhanced version of SDM, allows for handling small input cues and overcoming these pitfalls. Such cues were beyond the scope of original SDM work.

The ability of SDMSCue to overcome the input cue length constraint in the original SDM model provides superior functionality for associative memory. It allows for association and matching based on small hints or input cues.

The recall results obtained for SDMSCue are –in general- superior to those of original SDM. The gain achieved is quite significant statistically as well as objectively.

7. FUTURE RESEARCH

More comparisons and tests of SDMSCue vs. original SDM in specific AI and cognition domains may be done. Applying SDMSCue as an associative memory technique for some architectures as well as test beds; is under consideration.

BIBLIOGRAPHY

Anwar, Ashraf (1997) *TLCA: Traffic Light Control Agent.* Master Thesis, University of Memphis, TN, USA, Dec 1997.

Anwar, Ashraf (1999) *Sparse Distributed Memory with Evolutionary Mechanisms.* Proceedings of Genetic and Evolutionary Computation Conference Workshop (GECCO) 1999, p. 339-40.

Anwar, Ashraf, Dasgupta, Dipankar, and Franklin, Stan (1999) *Using Genetic Algorithms for Sparse Distributed Memory Initialization.* Proceedings of Congress on Evolutionary Computation (CEC99), Jul 1999.

Anwar, Ashraf, and Franklin, Stan (2003) *Sparse Distributed Memory for 鏠onscious Software Agents.* Cognitive Systems Research Journal, Dec 2003, v 4 n 4, p 339-54, UK: Elsevier.

Evans, Richard, and Surkan, Alvin. (1991) *Relating Number of Processing Elements in a Sparse Distributed Memory Model to Learning Rate and Generalization.* APL Quote Quad, Aug 1991 v 21 n 4, p 166.

Franklin, Stan. (1995) *Artificial Minds.* MIT Press.

Franklin, Stan. (1997) *Autonomous Agents as Embodied AI.* Cybernetics and Systems, special issue on Epistemological Issues in Embedded AI.

Franklin, Stan. (2001). *Automating Human Information Agents.* Practical Applications of Intelligent Agents, ed. Z. Chen, and L. C. Jain. Berlin: Springer-Verlag.

Franklin, Stan, and Graesser, Art. (1999*). A Software Agent Model of Consciousness.* Consciousness and Cognition v 8, p 285-305.

Franklin, Stan, Kelemen, Arpad, and McCauley, Lee. (1998) *IDA: A Cognitive Agent Architecture.* IEEE transactions on Systems, Man, and Cybernetics, 1998.

Glenberg, Arthur M. (1997) *What Memory is for?* Behavioral and Brain Sciences.

Hely, T. (1994) *The Sparse Distributed Memory: A Neurobiologically Plausible Memory Model?* Master's Thesis, Dept. of Artificial Intelligence, Edinburgh University.

Kanerva, Pentti, and Raugh, Michael. (1987) *Sparse Distributed Memory.* RIACS, Annual Report 1987, NASA Ames Research Center, Moffett Field, CA, USA.

Kanerva, Pentti. (1988a) *Sparse Distributed Memory.* MIT Press.

Kanerva, Pentti. (1988b) *The Organization of an Autonomous Learning System.* RIACS-TR-88, NASA Ames Research Center, Moffett Field, CA, USA.

Kanji, Gopal K. (1999) *100 Statistical Tests.* Sage Publications.

Karlsson, Roland. (1995) *Evaluation of a Fast Activation Mechanism for the Kanerva SDM.* RWCP Neuro SICS Laboratory.

Kosslyn, Stephen M., and Koenig, Olivier. (1992) *Wet Mind.* Macmillan Inc.

Kristoferson, Jan. (1995a) *Best Probability of Activation and Performance Comparisons for Several Designs of SDM.* RWCP Neuro SICS Laboratory.

Kristoferson, Jan. (1995b) *Some Comments on the Information Stored in SDM.* RWCP Neuro SICS Laboratory.

Loftus, Geoffrey, and Loftus, Elizabeth (1976) *Human Memory, The Processing of Information.* Lawrence Erlbaum Associates.

Rao, Rajesh P. N., and Fuentes, Olac. (1996) *Learning Navigational Behaviors using a Predictive Sparse Distributed Memory.* Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior, p 382.

Rao, Rajesh P. N., and Fuentes, Olac. (1998) *Hierarchical Learning of Navigation Behaviors* *in an Autonomous Robot using a Predictive Sparse Distributed Memory.* Machine Learning, Apr 1998 v31 n 1/3, p 87.

Rogers, D. (1988a) *Kanerva's Sparse Distributed Memory: An Associative Memory Algorithm Well-Suited to the Connection Machine.* International Journal of High Speed Computing, p 349.

Rogers, D. (1988b) *Using data tagging to improve the performance of Kanerva's Sparse Distributed Memory.* RIACS-TR-88.1, NASA Ames Research Center, Moffett Field, CA, USA.

Ryan, S., and Andreae, J. (1995) *Improving the Performance of Kanerva's Associative Memory.* IEEE Transactions on Neural Networks 6-1, p 125.

Scott, E., Fuller, C., and O'Brien, W. (1993) *Sparse Distributed Associative Memory for the Identification of Aerospace Acoustic Sources.* AIAA Journal, Sep 1993 v 31 n 9, p 1583.

Vogt, W. Paul (1998) *Dictionary of Statistics & Methodology, 2nd edition.* Sage Publications.

Willshaw, David. (1990) Coded Memories. A Commentary on 'Sparse Distributed Memory', by Pentti Kanerva. Cognitive Neuropsychology v 7 n 3, p 245.