Kanerva's Sparse Distributed Memory: An Object-Oriented Implementation on the Connection Machine

Andreas Turk*
German Research Center
for Artificial Intelligence
P. O. Box 20 80
D-6 76 08 Kaiserslautern
Germany

Günther Görz
Univ. of Erlangen-Nürnberg
IMMD (Computer Science) VIII — A. I.
Am Weichselgarten 9
D-9 10 58 Erlangen
Germany

Abstract

This paper reports on an implementation of Kanerva's *Sparse Distributed Memory* for the Connection Machine. In order to accomplish a modular and adaptive software library we applied a plain object-oriented programming style to the Common Lisp extension **lisp*. Some variations of the original model, the selected coordinate design, the hyperplane design, and a new general design, as well as the folded SDM due to Kanerva are realized. It has been necessary to elaborate a uniform presentation of the theoretical foundations the different designs are based on. We demonstrate the simulator's functionality with some simple applications. Runtime comparisons are given. We encourage the use of our simulation tool when outlining research topics of special interest to SDM.

1 Introduction

Uncertainty, vagueness and associativity have been a challenge for symbolic knowledge processing in AI. Apart from efforts to model these properties inside symbol-based AI, the development of hybrid systems by integrating AI methods with connectionist memory models can be of certain benefit for e. g. inference or natural language processing. The advantages of both approaches should be combined: symbolic AI makes use of explicit methods and knowledge and thus leads to a certain amount of comprehensibility, whereas neural networks are capable of modeling knowledge which is difficult to express explicitly. In addition they offer an easy way to apply case-inherent statistics. The connectionist paradigm, in particular, could gain new significance by this confluence.

One of the most clear and sound connectionist models is the *Sparse Distributed Memory*, as introduced by Pentti Kanerva [1988; 1992]. SDM is rooted in a mathematical idea complying with a convenient deduction of quantitative properties. It has also been described as a general *random access memory* (cf. [Kanerva, 1988], ch. 2), such that an easy hardware realization could be expected. On this basis a further description of SDM could be developed, using formal neurons.

From results of physiologically motivated brain research it has been argued that SDM models the physical structure of parts of the cerebellar cortex of mammals, as well as its functionality, the control of learned motor activity. SDM was designed to resemble some specific qualities of human cognition: dynamic recollection, distinct degrees of uncertainty in concept recognition, associativity in recalling memory contents, and the faculty for building abstractions.

SDM's technical properties, e. g. capacity, recognition rate, and convergence, have been investigated well. Now even the hardware restrictions, which for some time made the realization of SDM simulators of appropriate time efficiency impossible, have disappeared

We introduce a tool for experiments with SDM, running on a Connection Machine 2 [Hillis, 1985]. Compared to other simulators it has some specific advantages:

- There are in principle no hardware restrictions on the size of simulated SDMs, because data can be swapped between main and background memory of the CM 2.
- Data types are chosen in order to optimize execution time and reduce memory waste (sec. 3, 5).
- The software is engineered: it is modular, adaptive and extensible.
- Modules for the most important architectures of SDM-like memory models are predefined.

Flexible investigation tools are required to compare the behavior of differently designed SDMs. We found some attempts to develop simulators for SDM of which the closest one to our approach is that of Rogers [1989]. The main reason for providing an implementation of our own has been that all of them are neither capable to simulate SDMs of an appropriate size nor in a tolerable time [Matz, 1993].

2 The Memory Model

The SDM memory model is based on a small subset of the space $\{0,1\}^n$. Elements of this subset, the so-called *hard locations*, can be regarded as registers. The SDM is built up from three major parts concerning storage, the addressing mechanism and control (fig. 1): the *address*

*Previously at Univ. of Erlangen-Nürnberg.

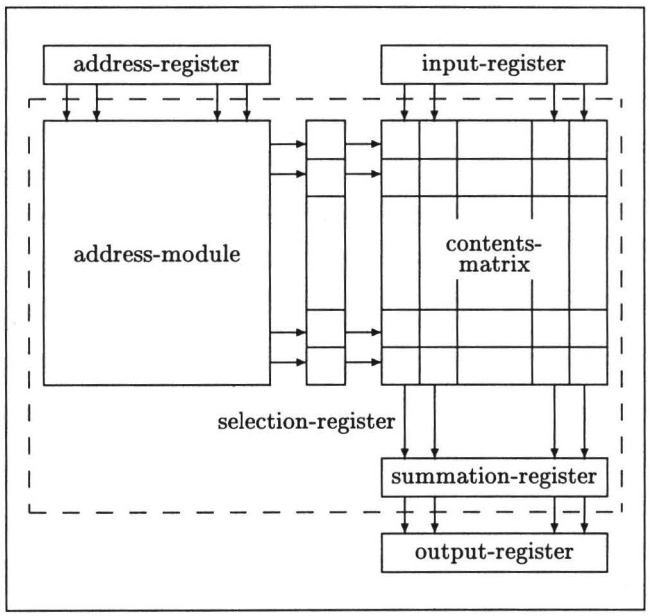

Figure 1: SDM as a register machine

module, containing address matrix and Hamming register, the *contents module* (selection register, contents matrix and summation register) and *external registers* for address, input and output. All of them employ binary counters, except the contents matrix which is built from integer counters. Counters can be considered as (weighted) connections in a neural-network-like representation of SDM. SDM's functionality is given by three operations: the *addressing* of hard locations, and the *writing* and *reading* of data. Addressing serves to mark a subset of registers (rows) of the contents matrix as active. The input register is added to the active memory locations by writing ("1"s cause the particular counters to increase, "0"s to decrease by one). The reading operation stores the column-wise sums of active registers in the summation register which in turn is matched against an external threshold. Resulting values are stored in the output register ("1" if greater than or equal to, "0" if less than the threshold).

With *selected coordinate design*, *hyperplane design* and several intermediate designs certain modifications of the original SDM have been accomplished [Jaeckel, 1989a; 1989b]. In principle these variants differ from Kanerva's basic model only in the methods of selecting sets of active registers. In *Kanerva's SDM* the address matrix is a uniform randomly initialized, constant binary matrix of dimensions $m \times n$. During addressing for each row of the address matrix the *Hamming distance* (defined below) to the address register has to be calculated. Locations that settle within a given radius are selected.

Choosing coordinates in $\{0,1\}^n$ corresponds to the selection of columns from the address matrix. The *selected coordinates SDM* provides, for each row of the address matrix, a different set of constant size containing randomly selected coordinates. Activation is effected for those locations that have matching values in their selected coordinates, when compared to the address register. This is to simulate a sparse matrix. The *intermediate design* also consists of sets of selected coordinates, but addressing is done as in basic SDM. Thus Kanerva's SDM and the selected coordinates SDM both can be specified from the more general intermediate design.

Another class of SDM is introduced by the *hyperplane design*. Addresses and data are assumed to contain "1s" with a probability of 0.1. This small proportion of active connections is physiologically motivated. As before, a set of selected coordinates exists for each register, but now this set is meant to enumerate exactly all activated connections ("1s" in the address-register). Thus all selected coordinates of activated locations must reference a "1" in the corresponding coordinate of the address register instead of matching "0"s and "1"s. This makes an explicit address matrix redundant.

The hyperplane design can be regarded as a counterpart to the selected coordinates design. A corresponding variant of the basic SDM is easily defined: the *adapted Kanerva design* expects a probability of 0.1 for neural activity in address matrix and data. No further changes are made. Finally the *intermediate hyperplane design* functions as a counterpart to the intermediate design comprising hyperplane and adapted Kanerva.

To reduce complexity we have defined the *generalized design*, comprising all of the SDM variants mentioned above. For this purpose it was necessary to formalize the different models in a uniform manner, so that all of them can be derived by an appropriate choice of parameters [Turk, 1993]:

Addressing operations serve to activate a precisely predictable number of registers in a reproducible way. Close addresses must effect a larger intersection of active registers than distant ones. Though Jaeckel [1989a; 1989b] based the measure on the logical **and**, we will here use measures[1] based on the logical **nand**, the *zero distance*

$$\mathrm{d}^{(n)}_{\mathrm{nand}}(x,y) = \sum_{i=0}^{n-1}(x_i \odot y_i), \text{ w. } a \odot b = \begin{cases} 0 & \text{if } a = b = 1 \\ 1 & \text{else} \end{cases}$$

or based on the logical **xor**, the *Hamming distance*, as used in the original SDM

$$\mathrm{d}^{(n)}_{\mathrm{xor}}(x,y) = \sum_{i=0}^{n-1}(x_i \oplus y_i), \text{ with } a \oplus b = \begin{cases} 1 & \text{if } a \neq b \\ 0 & \text{if } a = b \end{cases}$$

for the convenience of easy conversion. Arguments to these functions are memory items located in the space $\{0,1\}^n$.

A further modification has been applied: Since the combinatorial deduction tends to become rather complicated [Jaeckel, 1989a; 1989b], we preferred a statistical foundation. All probabilities are derived by application of the Binomial distribution or the Gaussian distribution as its approximation.

The new generalized design requires various parameters: n is the dimension of the address space, k the number of selected coordinates, p is the activation probability of a single register and h stands for the distance

[1] These measures need not define a metric on the $\{0,1\}^n$.

design	parameters				activation
	p_{one}	dist	k	h	
K	0.5	xor	n		$p(h, n, 0.5)$
SC	0.5	xor		0	$p(0, k, 0.5)$
I	0.5	xor			$p(h, k, 0.5)$
AK	0.1	xor	n		$p(h, n, 0.82)$
HP	0.1	nand		0	$p(0, k, 0.9)$
IH	0.1	nand			$p(h, k, 0.9)$

Table 1: Various SDM designs specified from the *generalized design*

of activation. In addition, a probability p_{one} for the occurrence of connections ("1s") in the address matrix has to be defined. All of them are mutually dependent and must be considered when the optimal probability of activation $p_{opt} = \frac{1}{\sqrt[3]{2mt}}$ is approximated. m represents the number of hard locations and t the hypothetical size of the set of data to be stored.[2] In brief, the cumulative Binomial distribution is given by

$$p(k, n, p) := P(X^{(n)} \leq k; p) = \sum_{i=0}^{k} P(X^{(n)} = k; p)$$

and the Binomial distribution can be defined as:

$$P(X^{(n)} = k; p) = \binom{n}{k} p^k q^{n-k}, q = 1-p, k \in \{0, \ldots, n\}$$

$X^{(n)}$ is meant to be a discrete random vector.

By choice of parameters we can specify various SDM designs from the most general one. Tab. 1 shows how the actual distance measure **dist** $\in$ {**xor**, **nand**} and the parameters k, h and p_{one} must be selected in order to obtain one of the models *Kanerva* (K), *selected coordinates* (SC), *intermediate* (I), *adapted Kanerva* (AK), *hyperplane* (HP), and *intermediate hyperplane* (IH). Actual activation probabilities are deduced. A combination of **dist** and p_{one} has to be considered to apply an appropriate value for p to the Binomial distribution. In addition arbitrary intermediate models are possible. A complete quantitative comparison is presented in [Turk, 1993]. Jaeckel [1989b] stated that SC and HP designs show a better selectivity than the K design and consequently have a higher storage capacity (up to five times). On the other hand their capability for associating concepts is, of course, reduced.

A storage problem which arises for overlapping sequences in the application of hetero-associative addressing is discussed by Kanerva [1988] (ch. 8). His solution assembles separate SDMs into a *folded SDM* of a definite depth performing an accurate indexing on those sequences. A k-folded SDM employs k *inner SDMs*, each one acting on individually delayed addresses. Sec. 5 gives an example.

[2]Compare [Kanerva, 1992; Jaeckel, 1989a; Chou, 1988] for a deduction.

3 Algorithms and Data Types

The CM 2 hardware architecture is almost optimally well suited to the requirements of SDM calculations. A large number of processors can be connected efficiently in the form of a virtual grid of arbitrary dimensions. Each processor is equipped with its own local storage. This allows for uniform operations on the elements of large matrices (*SIMD-parallelism*).

We decided to use the programming language **lisp*, a Common Lisp extension. **lisp* supports the adaption of parallel algorithms to the CM 2 architecture by special functions derived from standard Common Lisp functions and macros. They are marked by a preceding `*`, like `*set`, or by the suffix `!!`, like in `+!!`. Execution is restricted to all active CM 2 processors and takes place in parallel. Input to **lisp* functions is stored in so-called *pvars* (parallel variables) which are physically located in the storage of the particular CM 2 processors. Data transfer between a front-end computer and the CM 2 is provided.

Tab. 2 contains some variable declarations and sizes which are essential for a SDM simulator.[3]. Appropriate geometries of processor grids are 1 and 2. All variables are realized as *global pvars* instead of *permanent pvars* which would be the usual *pvar* type, to permit their binding to local identifiers — necessary to avoid name conflicts when simultaneously maintaining multiple SDMs. Type declarations are indispensable if time and space efficiency are to be increased by **lisp* compilation. The CM 2 requires the restriction of grid sizes to a power of two, also for reasons of efficiency.

The most important part of the code deals with addressing, writing, and reading, as pointed out in sec. 2. For addressing we distinguish two cases:[4]

1. Hamming distance

```
with predefined one-dimensional processor grid
of size m
  for each processor do in parallel
    l1 := logxor(address-register,
                 address-matrix)
    l2 := logand(coordinates, l1)
    hamming-register := logcount(l2)
    selection-register :=
      hamming register <= radius
```

   Each CM 2 processor has a single part of a *pvar* in its local storage, for instance one single row of the `address-matrix`. *Common Lisp variables* like `address-register` must be demanded from the **lisp* host machine.

2. Zero distance

   Only the first assignment has to be replaced:

```
l1 := lognot(address-register)
```

   As pointed out before (sec. 2) no address matrix is required to compute the zero distance.

[3]The maximum size of a SDM wrt. a specific CM 2 configuration is harder to find than expected: In **lisp* the temporary allocation of stack memory — e. g. during the calculation of return values of function calls — occurs automatically and cannot be influenced. Compare sec. 5 for particular instances.

[4]Instead of **lisp*-code we use a form of intuitive pseudocode for clarity. Transfer to **lisp* can easily be accomplished.

variable	data type	geometry
address-register input-register output-register	(unsigned-byte *register-length*)	
address-matrix coordinates	(unsigned-byte-pvar *register-length*)	m
selection-register	(boolean-pvar)	m
hamming-register	(unsigned-byte-pvar k)	m
selection-matrix	(boolean-pvar)	$n \times m$
contents-matrix	(signed-byte-pvar c)	$n \times m$
summation-register	(array (signed-byte *) 2)	

Table 2: Variables and data types

Addressing is executed row-wise in parallel. A different column-wise parallel handling of the contents matrix occurs during a write operation:

```
with predefined two-dimensional processor grid of
size n x m
  for each processor do in parallel
    when actual position in contents-matrix is
    selected
      case actual bit in input-register is
        0: add-matrix := -1
        1: add-matrix :=  1
    contents-matrix := contents-matrix + add-matrix
```

Register overflow must be prevented.

Since reading from memory involves a column-wise addition of activated counters, an appropriate parallelization is once more by columns:

```
with predefined two-dimensional processor grid of
size n x m
  for each processor do in parallel
    when actual position in contents-matrix is
    selected
      result := sum of all preceding values in
                contents-matrix wrt. second
                processor grid dimension
    summation-register :=
      highest activated row of contents-matrix
```

This so-called *spread sum* can easily be modeled by a combination of two **lisp* functions, `scan!!` and `+!!`.

**lisp* embodies an ideal tool to encode the parallel algorithms for the various memory models described in sec. 2. It functions as a mediator between abstract memory design and particular representations on the CM 2 hardware.

The appropriateness of **lisp* and the CM 2 was reported first by Rogers [1989], but our implementation differs from his in the choice of efficient data types (integers instead of floats), extended functionality (generalized design, cf. sec. 2), and an object-oriented framework which allows for an elegant combination of several SDMs in order to gain a larger or more complex memory model.

4 Library Structure and Encapsulation

The SDM simulator we introduce has been implemented in a plain object-oriented programming style based on flavors. It employs simple object classes, direct instances, and communication, but no further oo properties like multiple inheritance or sophisticated class hierarchies. Classes have been realized as generic functions which return a specific instance depending on their actual parameters. The resulting object is represented by a *closure* which contains all necessary methods and variables. Thus any number of SDMs can coexist in the front-end storage without influencing each other.[5] We have defined a SDM class by the generic function `create-sdm` which accepts parameters for distance measure, individual name, and values m, h, k, and p_{one} among others. As the addressing operation suffices to specify different SDM variants (sec. 2), furthermore a class for addressing strategies corresponding to the particular model is needed as an argument. Two such classes exist: one for addressing with the generalized SDM and another one containing optimized methods for Kanerva's base model.

SDM objects are able to dispatch messages for initialization, addressing, writing, and reading, as well as for adjustment of the activation radius and access to computational results. The handling of SDMs of unrestricted size is made possible by methods for swapping to and from the background memory. This includes file handling and memory allocation. Activation is initiated by the transfer of the simple messages `:store` and `:load`. Input, output, and address registers have to be handled externally. A typical session involves three steps: the *creation* of an SDM object by application of the corresponding class, *assignment* of values to the external registers and *communication* in order to initiate the desired operations. We have grouped all definitions concerning the simulation of simple SDMs together into the Common Lisp package **SDM**.

SDM models are characterized by the *compositionality principle*. It enables the division of a large SDM into parts in order to make a sequential processing possible. A vertical cut through the contents matrix and I/O registers could be carried out, if certain effect on common register selection were taken into account. Nevertheless we have decided to apply a horizontal cut through address and contents matrices in order to obtain smaller *complete* SDMs as parts. Taking the desired simulation

[5] In fact this was the reason why the use of an object-oriented programming style was adopted. The employment of an elaborated oo development tool, for instance the *Common Lisp Object System*, would be of certain benefit in future releases.

of folded SDMs into account this seems to be a more intuitive partitioning. Fig. 1 can now be viewed as one (of many) *inner SDM* combined with the external registers (dashed box). The particular SDM partitions are independent with one exception: reading out an overall sum requires one extra step to add up the inner summation registers.

In conjunction with the capability of swapping, memory partitioning allows for SDMs of arbitrary size. The package **BIG** contains functions necessary to handle sets of inner SDMs. Global *load-operate-store* cycles are built, managing step-by-step access to an array of single SDM objects. Thus it is possible to simulate SDMs of a size Kanerva [1988] stated as necessary to achieve certain theoretical properties (orthogonality of memory contents, convergence of sequential reading, memory capacity). The collection of different SDM types into a large *hybrid SDM* is also provided. This permits a SDM which employs different addressing methods at the same time, collecting sets[6] of similarly addressed registers, only if these sets are of a relevant size. Experiments in the combination of the advantages of different SDM models still remain to be carried out.

One further package (**FOLD**) provides the simulation of folded SDMs (sec. 2) using similar means of combination as above.

The usefulness of a SDM simulation depends very much on a correct value for the actual activation radius. Even small deviations from the optimal value may result in pathological states with all or none of the registers activated. The package **PROB** contains a collection of functions for an approximative calculation of the cumulative Binomial distribution of large numbers. They should be applied to calculate appropriate radii in advance.

A detailed description of functions as well as application examples can be found in [Turk, 1993].

5 Simulation Examples

The intended system behavior can be demonstrated by two simple examples. A few intuitive patterns have been predefined such that similarity can be seen immediately. We have made no attempt to meet statistical properties, e.g. parity of "1"s and "0"s, or to maximize orthogonality. The icons are called **caterpillar**, **pine** and the Roman numbers **one** to **six**.

The first experiment employs an SDM of type $n = 1,024$ dimensions, $m = 32,768$ hard locations, $k = 102$ selected coordinates, an activation radius of $h = 85$ and a probability of $p_{\text{one}} = 0.3$ for the appearance of "1"s in address matrix and data. The metric in use is defined by the zero distance. We added 20% of random noise to ten instances of the **caterpillar** icon. The resulting patterns are stored auto-associatively into the SDM:

[6]There is no order in the implementation of hard locations, since they are randomly selected in all SDM variants mentioned above.

Two gradual readings at a similarly noisy address:

results in the pattern:

What we obtained can be defined as the generation of an *abstraction*, since the original pattern **caterpillar** was in fact unknown.

The second experiment demonstrates the discrimination of sequences in a 4-folded SDM. The characteristics are $n = 1,024$ dimensions, $m = 8,192$ hard locations per fold, $k = 1,024$ selected coordinates, $h = 480$ as the activation radius and a probability for a single "1" of $p_{\text{one}} = 0.5$ in address matrix and data. We define the Hamming distance as the metric in use and hence are dealing with a SDM of type K. The hetero-associative loading of both sequences [**caterpillar**, **one**, **two**, **three**, **four**] and [**pine**, **one**, **two**, **three**, **five**] to an unfolded SDM results in inseparable memory contents. Reading four times gradually at the address **caterpillar** leads to nearly the same pattern as it does for the address **pine**: a combination of **four** and **five**. The 4-folded SDM we employ here, is capable of discriminating the above sequences correctly. This can be seen from the results obtained:

address: output:

address: output:

[Turk, 1993] provides complete traces.

The CM 2 at our disposal was equipped with 16 K one-bit processors, each with 32 K bit of local storage. Tab. 3 shows timing measurements of different SDM sizes wrt. *allocation*, *addressing*, *write-* and *read*-access. In the first row timing values for an SDM running on the **lisp simulator*, a CM 2 simulator for *Sparc stations*, can be found. Our experiences show that the **lisp simulator* is inadequate even for development purposes. Sorted by SDM magnitude, compiled ("comp") and non-compiled code is compared. It can be seen that compilation results in shorter execution times (up to a factor of five). Execution times of the front-end which runs the Common Lisp process, decrease even more (not contained). Meticulous declarations are necessary to provide the **lisp* compiler with sufficient information.[7]

Rogers [1989] tested his simulator with a K-type SDM of size $n = 256$ and $m = 8,192$. We obtained slightly

[7]We would have liked to present a comparison to the CM 5, but unfortunately we could not manage to get access to a CM 5 running **lisp*.

simulator	dimension	alloc.	address	write	read
*lisp (sim)	64 × 128	13.00 s	25.00 s	15.00 s	31.00 s
*lisp	1,024 × 8,192	0.19 s	0.14 s	1.01 s	0.82 s
*lisp (comp)	1,024 × 8,192	0.18 s	0.11 s	0.24 s	0.48 s
*lisp	256 × 8,192	0.14 s	0.05 s	0.25 s	0.20 s
*lisp (comp)	256 × 8,192	0.14 s	0.03 s	0.07 s	0.12 s
*lisp	64 × 8,192	0.12 s	0.02 s	0.13 s	0.05 s
*lisp (comp)	64 × 8,192	0.12 s	0.01 s	0.02 s	0.03 s
*lisp (comp)	64 × 32,768	0.14 s	0.04 s	0.06 s	0.12 s

Table 3: Runtime comparison of different SDM simulations

better results than the reported three write-read-cycles per second.

6 Related Work

Palm [1980] described another associative memory model, the *binary associative memory* (BAM). This model was designed to enable easy capacity estimations, but it also forms the basis of the PAN IV simulator (cf. [Palm, 1994]).[8] BAM can be compared directly with SDM, because its architecture is similar but simpler. BAMs consist of an initially empty binary contents matrix the counters of which can be only activated. No address decoding is performed, the address register indicates activation directly. Thus the number of hard locations is exactly the same as the dimension of the address space. BAM is an instance of SDM through *definite* activation of a subset of registers in combination with a slightly modified write operation.

Instead of the distributed representations in SDM, BAM stores exactly one representation for each association. As a result BAM has a higher storage capacity (which has been proved to be optimal) at the price of less robustness: permanent noise on input leads to an almost *completely* filled contents matrix during training phases. SDM instead manages to generate an abstraction, as demonstrated in sec. 5. BAM requires *sparse coding* even in normal use, whereas SDM input may or may not be coded sparsely (Kanerva vs. hyperplane designs). The performance of SDM subsumes BAM in the fields of e. g. pattern completion, pattern recognition, and the storage of sequences, but SDM possesses additional capabilities in building abstractions. The application of SDM to concrete programming problems is consequently more adequate than the use of BAM. It is a matter for further research to compare storage capacities under sparse coding for SDM.

In this context the *encoding problem* should be mentioned: the convenience of metrics like the Hamming distance does not come for free. Applications of connectionist memory models often involve an appropriate encoding of world knowledge. An isomorphic transition of relations of the conceptual domain into internal distances has to be found. The problem of variable binding that occurs, when neural processes are used to model logical inference, is an example as pointed out by Dorffner [1991]. Simple lexical coding has turned out to be insufficient. One can expect harder problems from the essential transfer of functional dependencies from world level to representational level, than from simple storage. This is caused by apparent incompatibilities between the necessity of unambiguous representations in certain cases and the desire for involving vagueness. The role of knowledge encoding will become a major research topic in the future. We suggest reducing similarity measurements to *contexts* to gain an extended comparability of data. Contexts are defined by subspaces in form of non-randomly selected coordinates. The effect is similiar to the use of different measures.

Encoding may also serve as an alternative way to handle overlapping sequences. Folding requires a set of separated SDMs, whereas an architecture-independent representation of sequences of arbitrary length is desirable. Encoding of system history as in [Jordan, 1986] combines the adequacy of one single unfolded SDM with a more plausible handling of overlapping sequences: long overlaps are harder to discriminate than short ones. Experiments on encoding are currently in preparation.

7 Future Work on the Implementation of Symbolic Structures with SDM

One of the major challenges facing connectionist approaches in typical computer science and artificial intelligence applications is the representation and processing of data structures, e. g. of (dynamic) sequences or sets. Fodor and Pylyshyn [1988] have argued that connectionist representations lack combinatorial syntactic and semantic structure. As Kanerva ([1988], ch. 8) pointed out, SDM is suited to store and retrieve sequences which in turn provide a basis for the implementation of complex data structures — as is common practice in Lisp programming. Therefore the next research step will be the specification and implementation of sequences, sets and temporal relations on top of the basic SDM storage model. With these means it will be possible to realize devices like finite automata. A particular challenge will be to provide the basic structures and operations for constraint-based natural language processing which has been a traditional domain for symbolic representations. Whereas most of those do not provide appropriate facilities to deal with vagueness and under-specification, these issues are a particular strength

[8] PAN IV is used in the WINA project — *Wissensverarbeitung in neuronaler Architektur* — at the Univ. of Ulm.

of SDM. Slack showed in a series of papers [1986; 1990] how to represent the basic structures and operations of a chart parser and of constraint-based grammar formalisms in a distributed memory model which closely resembles SDM. He even argues that, in doing so, it is not only possible to explain certain linguistic phenomena like unbounded dependency, but also that the use of distributed representations based on connectionist principles might influence theories developed at the level of symbolic representation. The investigation of the possibility of implementing a chart parser for context free grammars or even an augmentation with a constraint-based grammar formalism within SDM will not only provide valuable insights into bridging the gap between subsymbolic and symbolic representations, but also demonstrate a close integration of both approaches.

Acknowledgments

We are grateful to the *Gesellschaft für Mathematik und Datenverarbeitung* (GMD), Sankt Augustin, Germany, which provided access to the Connection Machine 2.

References

[Chou, 1988] P.A. Chou. The Capacity of the Kanerva Associative Memory is Exponential. Stanford University CA 94305, Stanford, USA, 1988.

[Dorffner, 1991] G. Dorffner. *Konnektionismus.* Teubner, Stuttgart, Germany, 1991.

[Fodor and Pylyshyn, 1988] J.A. Fodor and Z.W. Pylyshyn. Connectionism and Cognitive Architecture: A Critical Analysis. *Cognition*, 28:3–71, 1988.

[Hillis, 1985] W.D. Hillis. *The Connection Machine.* MIT Press, Cambridge, Massachusetts, 1985.

[Jaeckel, 1989a] Louis A. Jaeckel. An Alternative Design for a Sparse Distributed Memory. RIACS Technical Report 89.28, NASA Ames Research Center, Moffett Field, USA, July 1989.

[Jaeckel, 1989b] Louis A. Jaeckel. A Class of Designs for a Sparse Distributed Memory. RIACS Technical Report 89.30, NASA Ames Research Center, Moffett Field, USA, July 1989.

[Jordan, 1986] M. Jordan. Serial Order: A Parallel Distributed Processing Approach. ICS-UCSD, Report No. 8604, 1986.

[Kanerva, 1988] Pentti Kanerva. *Sparse Distributed Memory.* MIT Press, Cambridge, Massachusetts, 2 edition, 1988.

[Kanerva, 1992] Pentti Kanerva. Sparse Distributed Memory and Related Models. RIACS Technical Report 92.10, NASA Ames Research Center, Moffett Field, USA, April 1992.

[Matz, 1993] Karen Matz. SDM — Modell eines dünn besetzten, verteilten und hochgradig parallelen Speichers. FORWISS-Report, Bavarian Research Center for Knowledge-Based Systems, Erlangen, September 1993. FR-1993-010.

[Palm, 1980] G. Palm. On Associative Memory. *Biol. Cybern.*, 36:19–31, 1980.

[Palm, 1994] G. Palm. The PAN System and the WINA Project. Univ. of Ulm, Internal Memo, 1994.

[Rogers, 1989] David Rogers. Kanerva's Sparse Distributed Memory: An Associative Memory Algorithm Well-suited to the Connection Machine. *International Journal of High Speed Computing*, 1(2):349–365, 1989.

[Slack, 1986] J.M. Slack. Distributed Memory: A Basis for Chart Parsing. In *Proceedings of COLING-86*, pages 476–481, Bonn, 1986.

[Slack, 1990] J.M. Slack. Unbounded Dependency: Tying Strings to Rings. In *Proceedings of COLING-90*, pages 265–270, Helsinki, 1990.

[Turk, 1993] Andreas Turk. Parallele Implemetation eines verteilten assoziativen Speichers. IMMD 8, Univ. of Erlangen-Nürnberg, June 1993.