Binding and Normalization of Binary Sparse Distributed Representations by Context-Dependent Thinning

Dmitri A. Rachkovskij

V. M. Glushkov Cybernetics Center
Pr. Acad. Glushkova 40
Kiev 22, 252022
Ukraine

dar@infrm.kiev.ua
(*Preferable contact method*)

Ernst M. Kussul

Centro de Instrumentos
Universidad Nacional Autonoma de Mexico
Apartado Postal 70186
04510 Mexico D.F.
Mexico

ekussul@servidor.unam.mx

Paper draft (44 pages). Date: March 30, 1999.

Keywords: distributed representation, sparse coding, binary coding, binding, variable binding, representation of structure, structured representation, recursive representation, nested representation, compositional distributed representations, connectionist symbol processing,

Abstract

Distributed representations were often criticized as inappropriate for encoding of data with a complex structure. However Plate's Holographic Reduced Representations and Kanerva's Binary Spatter Codes are recent schemes that allow on-the-fly encoding of nested compositional structures by real-valued or dense binary vectors of fixed dimensionality.

In this paper we consider procedures of the Context-Dependent Thinning which were developed for representation of complex hierarchical items in the architecture of Associative-Projective Neural Networks. These procedures provide binding of items represented by sparse binary codevectors (with low probability of 1s). Such an encoding is biologically plausible and allows to reach high information capacity of distributed associative memory where the codevectors may be stored.

In distinction to known binding procedures, Context-Dependent Thinning allows to support the same low density (or sparseness) of the bound codevector for varied number of constituent codevectors. Besides, a bound codevector is not only similar to another one with similar constituent codevectors (as in other schemes), but it is also similar to the constituent codevectors themselves. This allows to estimate a structure similarity just by the overlap of codevectors, without the retrieval of the constituent codevectors. This also allows an easy retrieval of the constituent codevectors.

Examples of algorithmic and neural network implementations of the thinning procedures are considered. We also present representation examples of various types of nested structured data (propositions using role-filler and predicate-arguments representation, trees, directed acyclic graphs) using sparse codevectors of fixed dimension. Such representations may provide a fruitful alternative to the symbolic representations of traditional AI, as well as to the localist and microfeature-based connectionist representations.

1. Introduction

The problem of representing nested compositional structures is important for connectionist systems, because hierarchical structures are required for an adequate description of real world objects and situations.

In fully local representations, an item (entity, object) of any complexity level is represented by a single unit (node, neuron) (or set of units which has no common units with other items). Such representations are similar to symbolic ones and share their drawbacks. These drawbacks include the limitation of the number of representable items by the number of available units in the pool, and therefore, the impossibility to represent the combinatorial variety of real world objects. Besides, a unit corresponding to a complex object only represents its name and pointers to its components (constituents). Thus in order to determine the similarity of complex items, they should be unfolded into base-level (indecomposable) items.

The attractiveness of distributed representations was emphasized by the paradigm of cell assemblies (Hebb, 1949). In fully distributed representations, an item of any complexity level is represented by its configuration pattern over the whole pool of units. For binary units, this pattern is a subset of units which are in the active state. If subsets corresponding to various items intersect, then the number of these subsets is much more than the number of units in the pool, providing the opportunity to solve the problem of information capacity of representations. If similar items are represented by similar subsets of units, the degree of corresponding subsets' intersection could be the estimation of their similarity.

Potentially high information capacity of distributed representations gives hope to solve capacity problem for recursive structures, where more complex items are built from simpler ones. Representing composite items by the concatenation of activity patterns of their component items would increase the dimensionality of the coding pool. If the component items are encoded by pools of equal dimensionality, one could try to represent composite items as superposition of activity patterns of their components. The resulting coding pattern would have the same dimensionality.

However another problem arises here, known as "superposition catastrophe" (e.g. von der Malsburg, 1986) and "ghosts" or "false memory" in complement formulation (e.g. Feldman & Ballard, 1982; Hopfield, 1982). A simple example looks as follows. Let there be component items *A*, *B*, *C* and composite items *AB*, *AC*, *CB*. Let us represent any two of composite items, e.g. *AC* or *CB*. For this purpose, superimpose activity patterns corresponding to component items *A* and *C*, *C* and *B*. The ghost item *AB* also becomes represented in the result, though it is not needed. In the "superposition catastrophe" formulation, the problem consists in no way of telling which two items (*AB*, *AC*, or *AB*, *CB*, or *AC*, *CB*) make up the representation of composite pattern *ABC*, where the code patterns of all three component items are activated.

The supposition of no internal structure in distributed representations (or assemblies) (Legendy, 1970; von der Malsburg, 1986; Feldman, 1989) retarded their use for representation of complex data structures. The problem is to represent in the distributed fashion not only the information on the set of base-level components making up a complex hierarchical item, but also the information on the combinations in which they met, the grouping of those combinations, etc. That is, some mechanisms were needed for binding together the distributed representations of certain items at various hierarchical levels.

One of the approaches to binding is based on temporal synchronization of constituent activation (Milner, 1974; von der Malsburg, 1981, 1985; Shastri & Ajjanagadde, 1993).

Though this mechanism may be useful inside single level of composition, its capabilities to represent and store complex items with multiple levels of nesting are questionable (Hummel & Holyoak, 1997; Shastri & Ajjanagadde, 1993). Here we will consider binding mechanisms based on the activation of specific coding unit subsets corresponding to a group (combination) of items, the mechanisms that are closer to the so-called conjunctive coding approach (Smolensky, 1990; Hummel & Holyoak, 1997).

"Extra units" considered by Hinton (1981) represent various combinations of active units of two or more distributed patterns. The extra units can be considered as binding units encoding various combinations of distributedly encoded items by distinct distributed patterns. Such a representation of bound items is generally described by tensor products (Smolensky, 1990) and requires an exponential growth of the number of binding units with the number of bound items. However, as was already mentioned, for recursive structures it is desired that the dimensionality of patterns representing composite items is the same as of component items' patterns. Besides, the most important property of distributed representations - their similarity for similar structures - should be preserved.

The problem was discussed by Hinton (1990), and a number of mechanisms for construction of his "reduced descriptions" has been proposed. Hinton (1990), Pollack (1990), Sperduti (1994) get the reduced description of a composite pattern as a result of multilevel perceptron training using back-propagation algorithm. However, their patterns are low dimensional. Plate (1991, 1995) binds high dimensional patterns with gradual (real-valued) elements on the fly, without increase of the dimensionality, using the operation of circular convolution. Kanerva (1996) uses bitwise XOR to bind binary vectors with equal probability of 0s and 1s. Binary distributed representation are especially attractive, because binary bitwise operations are enough to handle them, providing the opportunity for significant simplification and acceleration of algorithmic implementations.

Sparse binary representations (with small fraction of 1s) are of special interest. The sparseness of codevectors allows to reach a high capacity of distributed associative memories (Willshaw, Buneman, & Longuet-Higgins, 1969; Palm, 1980; Lansner & Ekeberg, 1985) which can be used for their storage, and to accelerate still further its software and hardware implementations (e.g. Palm & Bonhoeffer, 1984; Kussul, Rachkovskij, & Baidyk, 1991a; Palm, 1993). Sparse encoding has also neurophysiological correlates (Foldiak & Young, 1995). The procedure for binding of sparse distributed representations ("normalization procedure") was proposed by Kussul as one of the features of Associative-Projective Neural Networks (Kussul, 1988, 1992; Kussul, Rachkovskij, & Baidyk, 1991). In this paper, we describe various versions of such a procedure, its possible neural network implementations, and provide examples of its use for the encoding of complex structures.

In section 2 we discuss representational problems encountered in the Associative-Projective Neural Networks and an approach for their solution. In section 3 the requirements on the Context-Dependent Thinning procedure for the binding and normalization of binary sparse codes are formulated. In section 4 several versions of the thinning procedure along with their algorithmic and neural network implementations are described. Some generalizations and notations are given in section 5. In section 6 retrieval of individual constituent codes from the combined item code is considered. In section 7 the similarity characteristics of the combined codes obtained as a result of Context-Dependent Thinning procedures are examined. In section 8 we show examples of encoding various structures using context-dependent thinning. Related work and general discussion is given in section 9, and conclusions are presented in section 10.

2. Representation of composite items in the APNN: the problems and the answer

2.1. Features of the APNN

The Associative-Projective Neural Networks (APNN) is the name of a neural network architecture proposed by Kussul in 1983 for the AI problems which require efficient manipulation with hierarchical information structures. Fragments of the architecture implemented in software and hardware were also used for solution of pattern recognition tasks (Kussul, 1992, 1993). APNN features of interest here are as follows (Kussul, 1992; Kussul, Rachkovskij, & Baidyk, 1991a; Amosov et al., 1991):

- Items of any complexity (an elementary feature, a relation, a complex structure, etc.) are represented by stochastic distributed activity patterns over the neuron field (pool of units);
- The neurons are binary and therefore patterns of activity are binary;
- Items of any complexity are represented over the neural fields of the same high dimensionality N;
- The number M of active neurons (units) in representation of items of various complexity is approximately (statistically) the same and small compared to the field dimensionality N. However M is large enough to maintain its own statistical stability.
- Items of various complexity level are memorized in different autoassociative neural networks with the same number N of neurons.

Thus, items of any complexity are encoded by a distributed stochastic sparse activity pattern of binary neurons in the neural fields of the same dimensionality N. It is convenient to represent activity patterns of neural fields as binary vectors, where 1s correspond to active neurons. Let us use bold-face letters for codevectors to distinguish them from the items they represent which are denoted by cursive letters. The number of 1s in $\mathbf{X}$ is denoted $|\mathbf{X}|$. We seek to make $|\mathbf{X}|=M$ for $\mathbf{X}$ of various complexity. The similarity of codevectors is determined by the number of 1s in their intersection or overlap: $|\mathbf{X} \wedge \mathbf{Y}|$, where $\wedge$ is elementwise conjunction of $\mathbf{X}$ and $\mathbf{Y}$. The probability of 1s in $\mathbf{X}$ (or density of 1s in $\mathbf{X}$, or simply the vector density) is $p(\mathbf{X})=|\mathbf{X}|/N$.

Information encoding by stochastic binary vectors with a small number of 1s allows to reach a high capacity of correlation-type neural network memory (also known as Hopfield network) using Hebbian learning rule (Wilshaw, Buneman, & Longuet-Higgins, 1969; Palm, 1980; Lansner & Ekeberg, 1985; Frolov & Muraviev, 1987, 1988; Frolov, 1989; Amari, 1989; Tsodyks, 1989). The codevectors we are talking about may be exemplified by vectors with M=100...1000, N=10 000...100 000. Though the maximal capacity is reached at M=logN (Willshaw, Buneman, & Longuet-Higgins, 1969; Palm, 1980), we use $M \approx \sqrt{N}$ to get a network with moderate number N of neurons and N^2 connections at sufficient statistical stability of M (e.g. the standard deviation of M less than 3%). Under this choice of the codevector density $p=M/N \approx 1/\sqrt{N}$, the information capacity holds high enough and the number of stored items can exceed the number of neurons in the network (Rachkovskij, 1990; Baidyk, Kussul, & Rachkovskij, 1990).

Let us consider the problems arising in the APNN and other neural network architectures with distributed representations when composite items are constructed.

2.2. Density of composite codevectors

The number H of component items (constituents) comprising a composite item grows exponentially with the nesting level, that is, with going to the higher levels of the part-whole hierarchy. If S items of the lower level l constitute an item of the adjacent higher level (l+1), then for the level (l+L) the number H becomes

$$H = S^L. \quad (2.1)$$

A simultaneous presence of several items comprising a composite item is encoded by the concurrent activation of their patterns, that is, by superposition of their codevectors. For binary vectors, superposition is bitwise disjunction. Therefore, construction of composite items will be accompanied by fast growth of density p' and respective number M' of 1s in their codevectors. For H different superimposed codevectors of density p:

$$p'_H = 1 - (1-p)^H \approx 1 - e^{-pH}, \quad (2.2)$$
$$M'_H = p'_H N. \quad (2.3)$$

Equation 2.3 takes into account the "absorption" of coincident 1s that prevents the exponential growth of their number versus the composition level L. However it is important that p'>>p (see Figure 1) and M' >> M. Since the dimensionality N of codevectors representing items of various complexity is the same, the size of corresponding auto-associative neural networks, where the codevectors are stored and recalled, is also the same. Therefore at M' >> M ≈ √N (at the higher levels of hierarchy) their capacity will decrease dramatically. To support the information capacity, M' should not substantially exceed M. However, due to the requirement of statistical stability, the number of 1s in the code can not be reduced significantly. Besides, the operation of auto-associative neural network memory is facilitated for the same number of 1s in codevectors. Thus it is necessary to support the number of 1s in the codevectors of complex items approximately equal to M. (Though the variation of M between distinct hierarchical levels by several times may be tolerable and even desirable).

These provide one of the reasons why composite items should be represented not by all 1s of their component codevectors, but only by their fraction approximately equal to M (i.e. only by some M representatives of active neurons encoding their components).

2.3. Ghost patterns and false memories

The well-known problem of ghost patterns or superposition catastrophe was mentioned in the Introduction. It consists in losing the information on the membership of component codevectors in particular composite codevector, when several composite codevectors are superimposed in its turn.

This problem is due to the essential property of superposition operation. The contribution of each member to their superposition does not depend on the contributions of other members. For superposition by elementwise disjunction, $\mathbf{A} \vee \mathbf{B}$ and $\mathbf{A} \vee \mathbf{C}$ are equally similar to **A**. The result of superposition of codevectors corresponding to various combinations of base-level components contains no other information except for the participating components of the base level. Therefore if common items are constituents of several composite items, then the composition of the latter can not be determined by the

intersection (overlap) of the resulting codevector with the base-level codevectors. For example, let the composite item consist of component items *A, B, C, D, E, F*. Then how could one determine that it really consists of the items *ABD, BCE, CAF,* if there may be also other items, such as *ABC, DEF,* etc. ? In the formulation of "false patterns", superposition of composite patterns **ABC** and **DEF** generates false patterns ("ghosts") **ABD, BCE, CAF,** etc.

The problem of introducing false assemblies or memories into a neural network (e.g. Kussul, 1980; Hopfield, Feinstein, & Palmer, 1983; Vedenov, 1987, 1988) has the same origin as the problem of ghosts. Training of an associative memory of matrix type is usually performed using some version of Hebbian learning rule implemented by superimposing in the weight matrix the outer products of memorized codevectors. For binary connections, e.g.

$$W_{ij}' = W_{ij} \vee X_i X_j, \quad (2.4)$$

where X_i and X_j are the states of the i-th and the j-th neurons when the memorized pattern **X** is presented (i.e. the values of the corresponding bits of codevector **X**), W_{ij} and W_{ij}' are the connection weights between the i-th and the j-th neurons before and after the training act respectively, $\vee$ is the sign of disjunction. When this learning rule is sequentially used to memorize several patterns of composite items including partially coinciding constituents, false assemblies may appear - that is, memorized composite patterns that were not presented to the network.

For example, when items *ABD, BCE, CAF* are memorized, the false assembly *ABC* is formed in the network (Figure 2A). Moreover, various two-item assemblies *AB*, *AD*, etc. are present, which were not explicitly presented for storing.

The problem of introducing false assemblies can be avoided if non-distributed associative memory is used, where the patterns are not superimposed when stored and each composite codevector is placed into a separate memory word. However the problem of false patterns or superposition catastrophe still persists.

2.4. An idea of the thinning procedure

A systematic use of distributed representations provides the prerequisite to solve both the problem of codevector density growth and the superposition catastrophe. The idea of solution consists in including into the representation of a composite item not full sets of active neurons (or 1s) encoding its component items, but only their subsets. If we choose the fraction of 1s from the codevector of each component item so that the number of 1s in the codevector of a composite item is equal to M, then the density of 1s will be preserved in codevectors of various complexity. For example, if A=3 items of the level l comprise an item of the level l+1, then M/3 of 1s should be preserved from the codevector of each l-th level item. Then the item of the level l+1 will have M of 1s. If two items of the level l+1 comprise the item of the level l+2, then M/2 of 1s should be preserved from each codevector of the level l+1. Since the number M of 1s in the codevectors of composite items of various complexity is maintained, potential information capacity of associative neural networks of higher hierarchical levels is maintained as well.

Thus the component items are represented in the codevector of the composite item in a reduced form - by a fraction of their 1s. The idea that the items of higher hierarchical levels ("floors") should contain their components in reduced, compressed, coarsen form is well-accepted among those concerned with diverse aspects of Artificial Intelligence research.

Reduced representation of component items in the codevector of composite item realized in the APNN may be relevant to "coarsen models" of Amosov (1968), "reduced descriptions" of Hinton (1990), and "conceptual chunking" of Halford, Wilson, & Phillips (1996).

Reduced representation of component items in the codevectors of composite items also allows to solve the superposition catastrophe. If the subset of 1s included in the codevector of a composite item from each of the component items depends on the composition of component items, then different subsets of 1s from each component item will be found in the codevectors of different composite items. For example, non-identical subsets of 1s will be incorporated into the codevectors of items *ABC* and *ACD* from the codevector of item *A*. Therefore the component items will be bound together by the subsets of 1s delegated to the codevector of composite item. It hinders the occurrence of false patterns and assemblies.

For the example from the Introduction, when composite items *AC* and *CB* are available concurrently, we will get the following code of the composite item: $\mathbf{A(C)} \vee \mathbf{C(A)} \vee \mathbf{C(B)} \vee \mathbf{B(C)}$, where **X(Y)** stands for the subset of 1s in the codevector **X** that becomes incorporated into the composite codevector given **Y** as the other constituent. Therefore if $\mathbf{A(C)} \neq \mathbf{A(B)}$, $\mathbf{B(C)} \neq \mathbf{B(A)}$, we do not observe the ghost pattern $\mathbf{A(B)} \vee \mathbf{B(A)}$ in the resultant codevector.

For the example of Figure 2A, where false assemblies have been emerged, they do not emerge under reduced representation of items (Figure 2B). Now interassembly connections are formed between different subsets of active neurons which have moderate intersection. Therefore the connectivity of assembly corresponding to the non-presented item *ABC* is low.

That the codevector of a composite item contains the subsets of 1s from the component items preserves the information on the availability of component items in the composite item. That the composition of each subset of 1s depends on the availability of other component items preserves the information on the combinations in which the component items occurred. That the codevector of a composite item has approximately the same number of 1s as its components allows to use the combinations of such composite codevectors for construction of still more complex items of the higher hierarchical level.

Thus an opportunity emerges to build up codevectors of items of varied composition level containing the information not only on the presence of their components, but on the structure of their combinations as well. It provides the possibility to estimate the similarity of complex structures without their unfolding but simply as overlap of their codevectors.

The ability to determine the similarity of complex structured items and to manipulate them without their expanding through the components is considered by many authors as very important for Artificial Intelligence (Kussul, 1992; Hinton, 1990; Plate, 1995, 1997).Originally the procedure reducing the sets of coding 1s of each item from the group which makes up a composite item with due regard to context of other items from the same group was named "normalization" (Kussul, 1988; Kussul & Baidyk, 1990; Kussul, 1992). That name emphasized the property to maintain the number of 1s in the codes of composite items equal to that of component items. It may be also called "Context Dependent Representatives Election" for choosing of context-dependent 1s to the higher-level codevector. However in this paper we will call it "Context Dependent Thinning" (CDT) by its action mechanism.

3. Requirements on the Context Dependent Thinning procedures

Let us summarize the requirements on the CDT procedures and on the characteristics of codevectors produced by them. The procedures should process sparse binary codevectors. An important case of input data to be thinned is superimposed codevectors of component items. The procedures should output the codevector of the composite item where the codevectors of component items are bound and which density is equal to the density of component codevectors. Let us also call the resulting (output) codevector as a "combined" or "thinned" codevector. The requirements may be expressed as follows.

3.1. Determinism

Repeated application of the CDT procedures to the same input should produce the same output.

3.2. Input and output representation of component items

3.2.1. Input representation

The procedure should process one, two, or several codevectors. One important case of an input codevector is superimposed codevectors of component items.

3.2.2. Output representation

The codevector of each input component item should be represented in the output codevector by a fraction of its 1s (or their reversible permutation).

3.2.3. Proportionality

The number of 1s representing the component items in the output combined codevector should be proportional to the number of 1s in the codevectors of the component items. If the number of 1s in **A** and **B** is the same, then the number of 1s from **A** and **B** in the combined codevector of the composite item *AB* should also be (approximately) the same.

3.3. Density control of the output codevectors

3.3.1. The CDT procedures should maintain (approximately) uniform low density of output codevectors (small number M' of 1s) under varied number of input codevectors and their correlation degree.

3.3.2. The CDT procedures should be able to control the number M' of 1s in output codevectors inside some range around M (the number of 1s in the codevectors of component items). For one important special case, M'=M.

3.4. Similarity

3.4.1. Unstructured similarity

An output codevector of the CDT procedures should be similar to the codevector of each component item (or to its reversible permutation). Fulfillment of this requirement follows from fulfillment of requirement 3.2.2. The combined codevector of item *AB* is similar to the codevectors of items *A* and *B*. If the densities of component codevectors are the same, the magnitude of similarity is the same (as follows from 3.2.3).

3.4.2. Similarity of component item representations at the output

Similarity of representations of the same constituent in various output codevectors should be the more, the more similar are the codevectors of other constituents of composite items. The representation of item *A* in the codevector of composite item *ABC* at the output of the CDT procedure should be more similar to its representation in the codevector of *ABD* than in the codevector of *AEF*.

3.4.3. Structured similarity

If two sets (collections) of component items are similar, their codevectors at the output of the CDT procedure should be similar as well. It follows from 3.4.2. If the codevectors of items *A* and *A*', *B* and *B*' are similar, then combined codevector of *AB* should be similar to the combined codevector of *A*'*B*'. Or, combined codevector of *ABC* should be similar to the combined codevector of *ABD*.

3.4.4. Binding

Representation of certain item in an output combined codevector should be different for different sets (collections) of component items. Representation of item *A* in the codevector of item *ABC* should be different from the representation of *A* in *ABD*. Thus the representation of *A* in the combined codevector contains the information on the other constituents of a composite item.

4. Versions of the Context Dependent Thinning procedures

Let us consider some versions of the CDT procedure, their properties and implementations.

4.1. Direct conjunctive thinning of two codevectors

As mentioned in the Introduction, the operation of conjunction is important for binding. Actually, binding is fixing of the simultaneous presence of constituents. The result of conjunction is 1 (true) if both operands are 1s. Therefore if the input 1 corresponds to presence of an item, then the output 1 corresponds to presence of both (all) conjunct items.

For a binary distributed encoding, the operation of conjunction is executed on all the elements (bits) of binary codevectors **X** and **Y**:

$$Z_i = X_i \wedge Y_i,\ i=1,...,N. \quad (4.1)$$

Here X_i and Y_i are the elements of multidimensional binary vectors to be bound, Z_i is the element of thinned and bound result **Z**.

4.1.1. Fulfillment of the CDT requirements

The requirement of determinism (section 3.1) holds for the conjunctive thinning procedure (equation 4.1). The requirement 3.2.1 is not met, since only two codevectors are bound. Overlapping 1s of **X** and **Y** go to **Z**, therefore the requirement 3.2.2 holds. Since equal number of 1s from **X** and **Y** enters into **Z** even if **X** and **Y** are of different density, the requirement 3.2.3 is not fulfilled in the general case.

For stochastic independent vectors **X** and **Y** the density of the resulting vector **Z** is:

$$p(\mathbf{Z}) = p(\mathbf{X})p(\mathbf{Y}) < \min(p(\mathbf{X}),p(\mathbf{Y})) < 1. \quad (4.2)$$

Here min() selects the smallest of its arguments. Let us note that for correlated **X** and **Y** the density of 1s in **Z** depends on the degree of their correlation. Thus p(**Z**) is maintained the same only for independent codevectors of constant density, and the requirement 3.3.1 is generally not met. Since p(**Z**) for sparse vectors is substantially lower than p(**X**) and p(**Y**), the requirement 3.3.2 is not met and recursive construction of bound codevectors is not supported (see also Sjodin, Kanerva, Bjorn, & Kristoferson, 1998). If the requirements of 3.4 on various kinds of similarities are applied to only two codevectors, they may be considered as holding.

Thus the operation of conjunctive thinning of two patterns does not meet the requirements on the CDT procedure that are mainly concerned with the number of bound items and with the density control of the combined output codevector. However this operation has been applied by us for the encoding of external information, in particular, for binding of distributed binary codevectors of feature item and its numerical value (Kussul & Baidyk, 1990; Rachkovskij & Fedoseyeva, 1990; Artykutsa et al., 1991; Kussul, Rachkovskij, & Baidyk, 1991a, 1991b). The density p of the codevectors of features and numerical values was chosen so that to provide a specified density p' of the resulting codevector (Table 1, K=2).

Table 1. The density p of K independent component codevectors chosen to provide a specified density p' of resulting codevectors produced by their conjunction.

K	2	3	4	5	6	7	8	9	10	11	12
p' = 0.001	0.032	0.100	0.178	0.251	0.316	0.373	0.422	0.464	0.501	0.534	0.562
p' = 0.010	0.100	0.215	0.316	0.398	0.464	0.518	0.562	0.599	0.631	0.658	0.681
p' = 0.015	0.122	0.247	0.350	0.432	0.497	0.549	0.592	0.627	0.657	0.683	0.705

4.1.2. A neural network implementation

One obvious neural network implementation of direct conjunctive thinning of two binary codevectors is shown in Figure 3. In the neural network terms, units are called "neurons", and their pools are called "neural fields". There are two input fields F_{in1} and F_{in2} and one output field F_{out} consisting of N binary neurons each. Each input field is connected with the output field by a bundle of projective (1:1) connections. "Synapses" of the neurons of the output field have weights of +1. Each neuron of the output field summarizes the excitations of its two

inputs (synapses). The output is determined by comparison of the excitation level with the threshold $\theta = 1.5$. An output neuron becomes activated if both corresponding neurons of the input fields are activated. Therefore it performs conjunction of its inputs. Thus the activity pattern of the output field corresponds to bit conjunction of patterns present in the input fields.

Direct conjunctive thinning of two binary distributed patterns, its properties and neural network implementation are straightforward. We presented it here to use in considerations of the CDT procedures of the following sections.

4.2. Direct conjunctive thinning of several codevectors

To bind more than two items, it is natural to generalize equation 4.1:

$$Z_i = \bigwedge_{s=1}^{S} X_i(s), \quad i = 1...N, \quad (4.3)$$

where S is the number of codevectors to be thinned. Though this operation allows to bind two or more codevectors, the requirement 3.2.1 is not fully met, because single vector is not allowed. The density of resulting codevector **Z** depends on the densities of **X**(s) and on their number S. Therefore to meet the requirement 3.3.1, the densities of **X**(s) should be chosen depending on the number of thinned codevectors. The requirement 3.3.2 also does not hold.

A neural network implementation may appear as in Figure 4. The number of the input fields may be chosen equal to the maximum number S_{max} of patterns to be bound, and the threshold of the output field F_{out} should be S_{max}-0.5. Originally all neurons of all fields are activated, and then the patterns corresponding to the codevectors of the constituents to be bound are input to some fields. Or, if all the fields are originally disactivated, then the threshold of Fout should be set to S-0.5.

We applied direct conjunctive thinning to encode positions of visual features in the two-dimensional retina. Three codevectors were bound (S=3): the codevector of a feature, the codevector of its X-coordinate, and the codevector of its Y-coordinate (unpublished work of 1991-1992 on recognition of handwritten digits, letters, and words in collaboration with Wacom, Japan). Also, this technique was used to encode words and word combinations for text processing (Rachkovskij, 1996). In so doing, the codevectors of letters comprising words were bound (S>10). The density of codevectors to be bound by thinning was chosen so that to provide a specified density of the resulting codevector (Table 1, K=3...12).

4.3. Paired conjunctive thinning of two or several items

The codevectors to be bound by direct conjunctive thinning are initially separated. Let us consider the case when the codevectors of constituents are superimposed by disjunction:

$$Z_i = \bigvee_{s=1}^{S} X_i(s), \quad i = 1,...,N. \quad (4.4)$$

Conjunction of a vector with itself produces the same vector: $Z_i \wedge Z_i = Z_i$. So let us modify **Z** by permutation of all its elements and make conjunction with the initial vector:

$$Z_i \wedge \tilde{Z_i} = Z_i', \quad i=1,...,N. \quad (4.5)$$

Here $\tilde{Z_i}$ is the i-th element of the permutated vector. The permutated vector is independent of the initial vector, therefore the density of the result is

$$p(\mathbf{Z}') = p(\mathbf{Z})p(\tilde{\mathbf{Z}}). \quad (4.6)$$

Let us consider the composition of the resulting vector:

$$\begin{aligned}
Z_i' &= (X_i(1) \vee X_i(2) \vee X_i(3) \vee ...) \wedge \tilde{Z_i} = \\
&= (X_i(1) \vee X_i(2) \vee X_i(3) \vee ...) \wedge (\tilde{X_i}(1) \vee \tilde{X_i}(2) \vee \tilde{X_i}(3) \vee ...) = \\
&= X_i(1) \wedge (\tilde{X_i}(1) \vee \tilde{X_i}(2) \vee \tilde{X_i}(3) \vee ...) \vee \\
&\quad X_i(2) \wedge (\tilde{X_i}(1) \vee \tilde{X_i}(2) \vee \tilde{X_i}(3) \vee ...) \vee \\
&\quad X_i(3) \wedge (\tilde{X_i}(1) \vee \tilde{X_i}(2) \vee \tilde{X_i}(3) \vee ...) \vee ...= \\
&= (X_i(1) \wedge \tilde{X_i}(1)) \vee (X_i(1) \wedge \tilde{X_i}(2)) \vee (X_i(1) \wedge \tilde{X_i}(3)) \vee ... \\
&\quad (X_i(2) \wedge \tilde{X_i}(1)) \vee (X_i(2) \wedge \tilde{X_i}(2)) \vee (X_i(2) \wedge \tilde{X_i}(3)) \vee ... \\
&\quad (X_i(3) \wedge \tilde{X_i}(1)) \vee (X_i(3) \wedge \tilde{X_i}(2)) \vee (X_i(3) \wedge \tilde{X_i}(3)) \vee ... \quad (4.7)
\end{aligned}$$

Thus the resulting codevector is the superposition of all possible pairs of bitwise codevector conjunctions, each pair including certain constituent codevector and certain permutated constituent codevector.

Because of initial disjunction of constituent codevectors, this procedure meets more requirements of section 3 than direct conjunction operations of sections 4.1 and 4.2. The requirement 3.2.1 is now fully satisfied. As follows from equation 4.7, each codevector of component item $\mathbf{X}(s)$ is thinned by conjunction with one and the same stochastic independent vector $\tilde{\mathbf{Z}}$. Therefore statistically the same fraction of 1s is left from each codevector $\mathbf{X}(s)$. Therefore the requirements 3.2.2 and 3.2.3 hold.

For S sparse codevectors of the same density $p(\mathbf{X}) << 1$

$$p(\mathbf{Z}) \approx Sp(\mathbf{X}), \quad (4.8)$$

$$p(\mathbf{Z}') = p(\mathbf{Z})p(\tilde{\mathbf{Z}}) = S^2p^2(\mathbf{X}). \quad (4.9)$$

To satisfy the requirement 3.3, $p(\mathbf{Z}')=p(\mathbf{X})$ should hold for various S. It means $p(\mathbf{X})$ should be equal to $1/S^2$. Therefore at fixed density $p(\mathbf{X})$ the requirement 3.3 is not satisfied for variable number S of component items. The similarity requirements 3.4 hold. In particular, the requirement 3.4.2 holds because the more is the number of identical items, the more identical conjunctions are superimposed in equation 4.7.

A neural network implementation is shown in Figure 5A. In distinction to Figure 3, the same pattern of superimposed component codevectors is activated in both input fields. The input neuron field F_{in1} is connected with the output field by the bundle of N direct projective connections. Each connection of this bundle connects neurons of the same number. The input neural field F_{in2} is connected with the output field by the bundle of N permutated projective connections. Each connection of this bundle connects neurons of different numbers.

Obviously, there is a lot of different configurations of permutated connections. Permutation by a shift is easy to implement in computer simulations (Figure 5B).

4.4. Additive CDT procedure

Though for paired conjunctive thinning the density of resulting codevector is closer to the density of each constituent codevector than for direct conjunction, it varies with the density of constituent codevectors and their number.

Let us make a codevector **Z** by dusjunction of codevectors of component items **X**(s). Since the density of constituent codevectors is low and their number is small, then the "absorption" of common 1s is low and, according to equations 4.8 and 4.9, p(**Z**) is approximately $1/S^2 p(\mathbf{X})$ times lower than p(**X**). For example, if p(**X**)=0.01, S=5 then p(**X**)/p(**Z**)≈4.

Therefore, to get p(**X**)≈p(**Z**'), let us superimpose appropriate number K of independent vectors with such a density:

$$\{Z_i\} = \bigvee_{k=1}^{K} (Z_i \wedge \tilde{Z_i}(k)) = Z_i \wedge (\bigvee_{k=1}^{K} \tilde{Z_i}(k)), \quad i = 1,...N. \quad (4.10)$$

Here {**Z**} is the output thinned vector, $\tilde{\mathbf{Z}}(k)$ is a unique (independent stochastic) permutation of elements of vector **Z**, fixed for each k.

The number K of vectors to be superimposed by disjunction can be determined as follows. To make the density of the thinned codevector equal to the density of its constituent codevector, the density of the superposition of permutated vectors should be:

$$p(\bigvee_{k=1}^{K} \tilde{Z}(k)) = 1/S. \quad (4.11)$$

Then after conjunction with **Z** (equation 4.10) we will get the needed density of {**Z**}:

$$p(\{\mathbf{Z}\}) = p(\mathbf{Z})/S = Sp(\mathbf{X})/S = p(\mathbf{X}). \quad (4.12)$$

Taking into account the "absorption" of 1s in disjunction of K permutated vectors $\tilde{\mathbf{Z}}$, equation (4.11) can be rewritten:

$$1/S = 1-(1-pS)^K. \quad (4.13)$$

Then

$$1-1/S = (1-pS)^K$$
$$\ln(1-1/S) = K \ln(1-pS)$$
$$K=\ln(1-1/S)/\ln(1-pS). \quad (4.14)$$

The dependence K(S) at different p is shown in Table 2.

Table 2. The number K of permutations of an input codevector needed to get the proper density of the output thinned codevector. (K should be rounded to the nearest integer). The

number of component codevectors superimposed in the input codevector is S, each having the density p. The additive version of Context-Dependent Thinning procedure.

Density p of component codevector	Number S of component codevectors in the input codevector					
	2	3	4	5	6	7
0.001	346.2	135.0	71.8	44.5	30.3	21.9
0.005	69.0	26.8	14.2	8.8	6.0	4.3
0.010	34.3	13.3	7.0	4.4	2.9	2.1

4.4.1. Meeting the requirements to CDT procedures

Since the configuration of each k-th permutation in equation 4.10 is fixed, the procedure of additive CDT is deterministic and the requirement 3.1 is met. The input vector **Z** to be thinned is the superposition of vectors encoding the component items. The number of these vectors may be variable, therefore the requirement 3.2.1 holds.

The output vector is obtained by conjunction of **Z** (or its invertible permutation) with the independent vector defined by equation 4.11. Therefore the 1s of all codevectors superimposed in **Z** are equally represented in {**Z**} and both 3.2.2 and 3.2.3 hold.

Density control of the output codevector for variable number of component items and density of their codevectors is realized by varying K (Table 2). Therefore the requirements 3.3 hold. Since the requirements 3.2.2 and 3.2.3 hold, the codevector {**Z**} is similar to the codevectors of all components **X**(s), and the requirement 3.4.1 holds.

The more similar are the components of one composite item to those of another, the more similar are their superimposed codevectors **Z**. Therefore the more similar are the vectors - disjunctions of K fixed permutations of **Z**. Therefore the more similar representations of each component will remain after conjunction (equation 4.10) with the codevector **Z**. Therefore the requirement 3.4.2 holds. Characteristics of this similarity will be considered in more detail in section 7.

Since different combinations of component items produce different **Z** and therefore different codevectors of K permutations of **Z**, representations of some component in the output codevector will be different for different combinations of component items (the requirement 3.4.4 holds) The more similar are representations of each component item in the output combined vector, the more similar are output codevectors (the requirement 3.4.3 holds).

4.4.2. An algorithmic implementation

As mentioned before, shift is an easily implementable permutation. Therefore an algorithmic implementation of this CDT procedure may be as in Figure 6A. Another example of this procedure does not require preliminary calculation of K (Figure 6B). In this version, conjunctions of the initial and permutated vectors are superimposed until the number of 1s in the output vector becomes equal to M.

4.4.3. A neural network implementation

A neural network implementation of the first example of the additive CDT procedure (Figure 6A) is shown in Figure 7.

There are four neural fields with the same number of neurons: two input fields F_{in1} and F_{in2}, the output field F_{out}, the intermediate field F_{int}. Both input fields contain the copies of the input vector **Z.** The neurons of the fields F_{in2} and F_{out} are connected by the bundle of direct projective connections (one-to-one). F_{int} and F_{out} are also connected in the same manner. However the neurons of F_{in1} and F_{out} are connected by K bundles of permutative projective connections. The threshold of F_{int} neurons is 0.5. Therefore F_{int} implements bit disjunction of K vectors, each being a unique permutation of bits of vector **Z**. The threshold of the output field F_{out} is equal to 1.5. Hence this field performs conjunction of the initial codevector **Z** from F_{in1} and the vector of K permutated and superimposed codevectors **Z** from F_{int}.

To choose K depending on the density of **Z**, the neural network implementation should incorporate some structures not shown in the figure. They should determine the density of the initial pattern **Z** and "activate" (turn on) K first bundles of permutative connections from their total number K_{max}. Alternatively, this structures should actuate the bundles of permutative connections one-by-one in the fixed order until the density of the output vector in F_{out} becomes M/N. Let us remind that bundles of shift permutative connections are used in algorithmic implementations.

4.5. Subtractive CDT procedure

Let us consider another version of the CDT procedure. Let us obtain {**Z**} by the additive CDT procedure as indicated above. Then let us take as an output vector the remaining 1s of **Z**. The resulting vector

$$\begin{aligned}\{Z_i\}' &= Z_i - \{Z_i\} = Zi - Zi \wedge (\tilde{Z_i}(1) \vee \tilde{Z_i}(2) \vee ... \vee \tilde{Z_i}(K)) = \\ &= Z_i \wedge (1 - (\tilde{Z_i}(1) \vee \tilde{Z_i}(2) \vee ... \vee \tilde{Z_i}(K))) = \\ &= Z_i \wedge \neg(\tilde{Z_i}(1) \vee \tilde{Z_i}(2) \vee ... \vee \tilde{Z_i}(K)) = \\ &= Z_i \wedge \neg\tilde{Z_i}(1) \wedge \neg\tilde{Z_i}(2) \wedge ... \wedge \neg\tilde{Z_i}(K). \quad (4.15)\end{aligned}$$

If we choose K to make the number of 1s in {**Z**}' equal to M, then this procedure will satisfy the requirements of section 3. Since the inversion of $\vee\tilde{Z}(k)$ is used in this procedure, its density before inversion should be 1-1/S (compare to equation 4.13). Therefore the number K of permutated vectors to be superimposed in order to obtain the required density (taking into account "absorption" of 1s) is determined from:

$$1\text{-}1/S = 1 - (1\text{-}pS)^K. \quad (4.16)$$

Then

$$K \approx \ln S/pS. \quad (4.17)$$

Algorithmic implementations of this subtractive CDT procedure (Kussul, 1988; Kussul & Baidyk, 1990; Kussul, 1992) are analogous to those presented for the additive CDT procedure in section 4.3.2.

Let us consider a neural network implementation of this procedure (Figure 8). There are three neuron fields of the same number of neurons: two input fields F_{in1} and F_{in2}, as well as the output field F_{out}. The copies of the input vector **Z** are in both input fields. The neurons of F_{in2} and F_{out} are connected by the bundle of direct projective connections (one-to-one). The neurons of F_{in1} and F_{out} are connected by K bundles of independent permutative connections. Unlike Figure 7, the synapses of permutated connections are inhibitory (the weight is -1). The threshold of the output field neurons is 0.5. Therefore the neurons of **Z** remaining active in F_{out} are those non-inhibited by even if one of the permutative connections coming from the active neurons of **Z**.

Since the value of ln(S)/S is approximately the same at S=2,3,4,5 (Table 3, first row), one can choose K for a specified density of constituent codevectors p(**X**):

$$K \approx 0.34/p(\mathbf{X}). \quad (4.18)$$

At such K and S, $p(\{\mathbf{Z}\}') \approx p(\mathbf{X})$. Therefore the number K of permutative connection bundles can be fixed, and their ordered activation is not needed. So each neuron of F_{out} may be considered as connected with an average of K randomly chosen neurons of F_{in1} by inhibitory connections. More precise values of K (obtained as exact solution of equation 4.16) for different values of p are presented in Table 3 (rows 2-4).

Table 3. The function lnS/S and the number of permutations K of an input codevector needed to get the proper density of the output thinned codevector. (K should be rounded to the nearest integer). The number of component codevectors superimposed in the input codevector is S, each having the density p. The subtractive version of Context-Dependent Thinning procedure.

	Number S of component codevectors in the input codevector					
	2	3	4	5	6	7
lnS/S	0.347	0.366	0.347	0.322	0.299	0.278
p=0.001	346.2	365.7	345.9	321.1	297.7	277.0
p=0.005	69.0	72.7	68.6	63.6	58.8	54.6
p=0.010	34.3	36.1	34.0	31.4	29.0	26.8

It has been just this version of the CDT procedure which has been proposed as "normalization procedure" (Kussul, 1988; Kussul & Baidyk, 1990; Amosov et al., 1991). We have used it in the multilevel APNN for sequence processing (Rachkovskij, 1990b; Kussul & Rachkovskij, 1991). We have also used it for binding of sparse codes in perceptron-like classifiers (Kussul, Baidyk, Lukovich, & Rachkovskij, 1993) and in one-level APNN applied for recognition of vowels (Rachkovskij & Fedoseyeva, 1990, 1991), textures (Artykutsa et al., 1991; Kussul, Rachkovskij, & Baidyk, 1991b), shapes (Kussul & Baidyk, 1990), handprinted characters (Lavrenyuk, 1995), logical inference (Kasatkin & Kasatkina, 1991).

5. Procedures of auto-, hetero-, selfless thinning and notation

In sections 4.3-4.5 we considered the versions of thinning procedures where a single vector (superposition of constituent codevectors) was the input. The corresponding pattern of

activity was present both in the field F_{in1} and F_{in2} (Figures 5A, 7, 8), and therefore the input vector thinned itself. Let us call these procedures "auto-thinning" or "auto-CDT" and denote them as

$^{\$}(\mathbf{U})$. (5.1)

Here **U** is the codevector to be thinned (usually superposition of constituent codevectors) which is in the input fields F_{in1} and F_{in2} of Figures 5A, 7, 8; $ is the configuration label of permutative connections. A unique pattern of bundles of permutative connections corresponds to each label $. Therefore differently labeled CDT procedures implement different thinning. For example, different labels $ will use different seeds in the algorithmic implementations (Figure 6). Instead of various labels $, various bracket types can be used: {},[],<>,(),//,\\, etc. Unless otherwise specified, it is assumed that the number K of bundles is chosen to maintain the preset density of the thinned vector $\{\mathbf{U}\}$, usually $|\{\mathbf{U}\}|$=M.

It is possible to thin one codevector by another one if the pattern to be thinned is activated in F_{in1}, and the pattern which thins is activated in F_{in2}. Let us call such procedure hetero-CDT, hetero-thinning, thinning **U** from **W**. We denote hetero-thinning as

$^{\$}(\mathbf{U})_{\mathbf{W}}$. (5.2)

Here **W** is the pattern from which the thinning is performed (which thins). It is activated in F_{in2} of Figures 5A, 7, 8. **U** is the pattern which is thinned. It is activated in F_{in1}. $ is the configuration label of permutative connections. For auto-thinning, we may write $^{\$}(\mathbf{U}) = {}^{\$}(\mathbf{U})_{\mathbf{U}}$. Usually the codevector **U** to be thinned is a bit disjunction of constituent codevectors: $U_i = A_i \vee B_i \vee C_i \vee ...$, i=1,...,N. Let us denote it as $\mathbf{U} = \mathbf{A} \vee \mathbf{B} \vee \mathbf{C} \vee ...$ or simply **U**=**ABC**...

Examples:

Auto-thinning of composite item with components A, B, C using the pattern of permutative connections defined by the label "$" or {}:

$^{\$}(\mathbf{A}\vee\mathbf{B}\vee\mathbf{C})_{\mathbf{A}\vee\mathbf{B}\vee\mathbf{C}} = \{\mathbf{A}\vee\mathbf{B}\vee\mathbf{C}\} = {}^{\$}(\mathbf{ABC})_{\mathbf{ABC}} = \{\mathbf{ABC}\}$.

Hetero-thinning of composite item with components A,B,C from the component item C using the pattern of permutative connections defined by the label {} or $:

$^{\$}(\mathbf{A}\vee\mathbf{B}\vee\mathbf{C})_{\mathbf{C}} = \{\mathbf{A}\vee\mathbf{B}\vee\mathbf{C}\}_{\mathbf{C}} = {}^{\$}(\mathbf{ABC})_{\mathbf{C}}=\{\mathbf{ABC}\}_{\mathbf{C}}$.

For both additive and subtractive CDT procedures:

$\{\mathbf{ABC}\} = \{\mathbf{A}\}_{\mathbf{ABC}} \vee \{\mathbf{B}\}_{\mathbf{ABC}} \vee \{\mathbf{C}\}_{\mathbf{ABC}}$.

Analogous expression can be written for a composite pattern with another number of components.

For the additive CDT procedure it is also true:

$\{\mathbf{ABC}\} = \{\mathbf{A}\}_{\mathbf{ABC}} \vee \{\mathbf{B}\}_{\mathbf{ABC}} \vee \{\mathbf{C}\}_{\mathbf{ABC}} =$

$= \{\mathbf{A}\}_{\mathbf{A}} \vee \{\mathbf{B}\}_{\mathbf{A}} \vee \{\mathbf{C}\}_{\mathbf{A}} \vee \{\mathbf{A}\}_{\mathbf{B}} \vee \{\mathbf{B}\}_{\mathbf{B}} \vee \{\mathbf{C}\}_{\mathbf{B}} \vee \{\mathbf{A}\}_{\mathbf{C}} \vee \{\mathbf{B}\}_{\mathbf{C}} \vee \{\mathbf{C}\}_{\mathbf{C}}.$

For the subtractive CDT procedure we can write:

$\{\mathbf{A}\}_{\mathrm{ABC}} = \{\{\{\mathbf{A}\}_{\mathbf{A}}\}_{\mathbf{B}}\}_{\mathbf{C}}.$

Let us also consider a modification of the auto-CDT procedures which will be used in section 7.2. If we eliminate the thinning of a constituent codevector from itself, we obtain "selfless" auto-thinning. Let us denote it as $\{\mathbf{ABC}\}_{\backslash\mathbf{ABC}}$:

$\{\mathbf{ABC}\}_{\backslash\mathbf{ABC}} = \{\mathbf{A}\}_{\mathbf{BC}} \vee \{\mathbf{B}\}_{\mathbf{AC}} \vee \{\mathbf{C}\}_{\mathbf{AB}}.$

6. Retrieval of constituent codevectors

After thinning, the codevectors of constituents are present in the combined codevector of a composite item in a thinned, reduced form. We must be able to retrieve complete constituent codevectors. Since the requirement of the unstructured similarity 3.4.1 holds, the combined codevector is similar to its constituent codevectors. So if we have a full set (alphabet) of item codevectors of the preceding (lower) level of composition (hierarchy), we can compare the combined codevector to them. The similarity degree is determined by the overlap of codevectors. The alphabet items corresponding to the codevectors with maximum overlaps are the constituents of the composite item under consideration.

The search of the most similar constituents can be performed by a sequential comparison of the codevector to be decoded with all codevectors of the alphabet. An associative memory can be used to implement this operation in parallel. After retrieving of the full-sized constituent codevectors of the lower hierarchical level, one can then retrieve their constituent codevectors of still lower hierarchical level in an analogous way. For this purpose, their alphabet should be known as well. If the order of constituent retrieval is important, some auxiliary procedures can be used (Kussul, 1988; Amosov et al., 1991; Rachkovskij, 1990b; Kussul & Rachkovskij, 1991).

An example. Let us consider the alphabet of six component items A, B, C, D, E, F. They are encoded by stochastic fixed vectors of 100000 bits with approximately 1000 bits set to 1. Let us obtain the combined codevector $\{\mathbf{ABC}\}$ by the CDT procedure. The number of 1s in $\{\mathbf{ABC}\}$ in our numerical example is $|\{\mathbf{ABC}\}| = 1002$. Let us find the overlap of each component codevector with the combined codevector: $|\mathbf{A} \wedge \{\mathbf{ABC}\}| = 341$; $|\mathbf{B} \wedge \{\mathbf{ABC}\}| = 350$; $|\mathbf{C} \wedge \{\mathbf{ABC}\}| = 334$; $|\mathbf{D} \wedge \{\mathbf{ABC}\}| = 12$; $|\mathbf{E} \wedge \{\mathbf{ABC}\}| = 7$; $|\mathbf{F} \wedge \{\mathbf{ABC}\}| = 16$. So the representation of the component items A,B,C is substantially higher than the representation of the items D,E,F occurring due to a stochastic overlap of independent binary codevectors.

7. Similarity preservation by the thinning procedures

In this section, let us consider the similarity of thinned codevectors of composite items; as well as the similarity of thinned representations of component items in the thinned codevectors. These kinds of similarity are considered under different combinations of component items and different versions of thinning procedures.

Let us use the following Context-Dependent Thinning procedures:
- paired conjunctive thinning, section 4.3 (Paired-M);
- additive auto-CDT, section 4.4 (CDTadd);
- additive selfless auto-CDT (CDTadd-sl);
- subtractive auto-CDT, section 4.5 (CDTsub);
- subtractive selfless auto-CDT (CDTsub-sl)

For this experiment, let us first obtain the following codevectors by disjunctive superposition of the codevectors of their items:
Two-component codevectors **AB**, **AC**, **CD**;
Three-component codevectors **ABC**, **ABD**, **ADE**, **DEF**;
Four-component codevectors **ABCD**, **ABCE**, **ABEF**, **AEFG**, **EFGH**;
Five-component codevectors **ABCDE**, **ABCDF**, **ABCFG**, **ABFGH**, **AFGHI**, **FGHIJ.**

Then, for each thinning procedure, let us thin the codevectors of composite item down to the density of their constituent codevectors. (For paired conjunctive thinning, the density of component codevectors was chosen to get approximately M of 1s in the result).

7.1. Similarity of thinned codevectors

Let us find an overlap of various combined codevectors:
{**AB**} and {**AB**}, {**AC**}, {**CD**};
{**ABC**} and {**ABC**},{**ABD**}, {**ADE**}, {**DEF**};
{**ABCD**} and {**ABCD**}, {**ABCE**}, {**ABEF**}, {**AEFG**}, {**EFGH**};
{**ABCDE**} and {**ABCDE**}, {**ABCDF**}, {**ABCFG**}, {**ABFGH**}, {**AFGHI**}, {**FGHIJ**}.
Here {} is used to denote any thinning procedure. The overlap of vectors is normalized to 1.0, so that for codevectors **X** and **Y** it is determined as $|\mathbf{X} \wedge \mathbf{Y}|/|\mathbf{X}|$.

The results are shown in Figure 9. It can be seen that the overlap of combined codes for various versions of the CDT procedure is approximately equal to the square of the similarity of composite items. For example, the similarity of items AB and AC is 0.5 (one common item of two total), and the similarity (overlap) of their combined codevectors thinned by the selfless versions of both additive and subtractive CDT procedures is approximately 0.25.

7.2. Similarity of thinned representations of constituent codevectors

The value of an overlap of representations of item A in each thinned vector is obtained as follows. For two-component items, let us obtain the vector **X** consisting of 1s that coincide in the component vector **A** and in the thinned vector {**AB**}: $\mathbf{X}=\mathbf{A} \wedge \{\mathbf{AB}\}$. Analogously, $\mathbf{Y}=\mathbf{A} \wedge \{\mathbf{AC}\}$. Then the normalized value of their overlap is obtained as $|\mathbf{X} \wedge \mathbf{Y}|/|\mathbf{X}|$.

The same calculations for e.g. four-component items are as follows: $\mathbf{Q}=\mathbf{A} \wedge \{\mathbf{ABCD}\}$, $\mathbf{R}=\mathbf{A} \wedge \{\mathbf{ABCE}\}$, $\mathbf{S}=\mathbf{A} \wedge \{\mathbf{ABEF}\}$, $\mathbf{T}=\mathbf{A} \wedge \{\mathbf{AEFG}\}$, $\mathbf{U}=\mathbf{A} \wedge \{\mathbf{EFGH}\}$.The normalized values of overlaps are $|\mathbf{Q} \wedge \mathbf{R}|/|\mathbf{Q}|$, $|\mathbf{Q} \wedge \mathbf{S}|/|\mathbf{Q}|$, $|\mathbf{Q} \wedge \mathbf{T}|/|\mathbf{Q}|$, $|\mathbf{Q} \wedge \mathbf{U}|/|\mathbf{Q}|$.

The results are shown in Figure 10. It can be seen that the dependencies of representation similarity are different for these procedures: for the CDTadd and the CDTadd-sl they are close to linear, but for the CDTsub and CDTsub-sl they are close to bell-like. Which is preferable, should be determined by an application.

7.3. The influence of the "depth" of thinning

By the "depth" of thinning we understand the density value of a thinned codevector. Usually, we make it equal to the density of constituent codevectors. Here we vary the density of the thinned codevectors. The results are presented in Figure 11. We have used the composite items of five constituents with the additive and the subtractive CDT procedures. Therefore 5M 1s was in the input vector before thinning. We varied the number of 1s in the thinned codevectors from 4M to M/4.

The similarity of thinned codevectors is shown in Figure 11A. For a "shallow" thinning, where the resulting density is chosen to be near the density of an input vector (which consists of superimposed codevectors of constituents), the similarity degree of resulting vectors is close to that of input codevectors (the curve is close to linear). For a "deep" thinning, where the density of a thinned codevector is much less than the density of an input codevector, the similarity function behave as a power function, transforming from linear through quadratic to cubic (for the case of subtractive thinning).

The similarity of constituent representations in the thinned codevector is shown in Figure 11B. For the additive CDT procedure, the similarity function is linear, and its angle reaches approximately 45° for “deep” thinning. For the subtractive CDT procedure, the function is similar to the additive one for the “shallow” thinning, and becomes near-quadratic for the “deep” thinning.

8. Representation of structured expressions

Let us consider representation of various kinds of structured data by binary sparse codevectors of fixed dimensionality. In the examples below, the letters or labels denote items of the base level for a given expression. In its turn, they may also represent complex structured data.

8.1. Transformation of symbolic bracketed expressions into representations by codevectors

Performing the CDT procedure can be viewed as analog of introducing brackets into symbolic descriptions for correct account of syntax. As mentioned in section 5, the CDT procedures with different patterns of permutative connections are denoted by different types of brackets.

Therefore in order to represent a complex symbolic structure by a distributed binary codevector, one should

- map each symbol of the base-level item to the corresponding binary sparse codevector of fixed dimensionality;
- replace conventional brackets in symbolic bracketed representation by "thinning" ones. Each compositional level has its own "thinning" brackets, that is, configuration of permutative connections;
- superimpose the codevectors inside the brackets of the deepest nesting level by elementwise disjunction;
- perform the CDT procedure on superimposed codevectors using configuration of permutative connections corresponding to particular kind of "thinning" brackets;
- superimpose the resulting thinned vectors inside the brackets of the next nesting level;
- perform the CDT procedure on superimposed codevectors using an appropriate configuration of permutative connections corresponding to the brackets of that nesting level;
- repeat two previous points until the whole structure is encoded.

8.2. Representation of ordered items

For many propositions, the order of arguments is essential. To represent the order of items encoded by the codevectors, binding with appropriate roles is usually used.

One approach is to use explicit binding of role codevectors (agent-object, antecedent-consequent, or just ordinal number) with the item (filler) codevector. This binding can be realized by an auto- or hetero-CDT procedure (Rachkovskij, 1990b). The item A which is #3 may be represented as $\{\mathbf{A} \vee \mathbf{n3}\}$ or $\{\mathbf{A}\}_{\mathbf{n3}}$, where **n3** is the codevector of the "third place" role.

Another approach is to use implicit binding by providing different locations for different positions of an item in a proposition. To preserve fixed dimensionality of the codevectors, it was proposed to encode different positions by specific shifts of codevectors (Kussul & Baidyk, 1993). For our example we have **A**>>n3, where ">>" denotes shift, n3 is the number of 1-bit shifts corresponding to the "third place" of an item.

These and other techniques to represent the order of items have their pros and cons. Thus a specific technique should be chosen depending on the application. So we will not consider details here. It is important that such techniques exist, and we will denote a codevector of the item A at the n-th place simply by **A**_n.

Let us note that generally the modification of an item codevector to encode its ordinal number should be different for different nesting levels. It is analogous to having its own patterns of permutations for thinning at each level of nesting. Therefore the codevectors **A** and **B** should be modified in the same manner in {...**A**_n...} and {...**B**_n...}, but the codevector **A** should generally be modified differently in [...A_n...].

8.3. Examples

8.3.1. Role-filler structure

Representations of structures or propositions by the role-filler scheme are widely used (Smolensky, 1990; Pollack, 1990; Plate, 1991, 1995; Kanerva, 1996; Sperduti, 1994). Let us consider the relational instance

knows(Sam, loves(John, Mary)). (8.1)

Using the Holographic Reduced Representations, Plate (in press) represents it as:

L1 = love + loveagt∗john + loveobj∗mary, (8.2)
L2 = know + knowagt∗sam + knowobj∗L1, (8.3)

where ∗ stands for binding operation, and + denotes addition. In our representation:

L1 = [**love** ∨ {**loveagt** ∨ **john**} ∨ {**loveobj** ∨ **mary**}], (8.4)
L2 = /know ∨ **<knowagt** ∨ **sam>** ∨ **<knowobj** ∨ **L1>/**. (8.5)

8.3.2. Predicate-arguments structure

Let us consider representation of relational instances loves(John,Mary) and loves(Tom,Wendy) by the predicate-arguments (or symbol-argument-argument) structure (Halford, Wilson, & Phillips, in press):

loves∗**John**∗**Mary + loves**∗**Tom**∗**Wendy.** (8.6)

Using our representation, we obtain:

[{**loves**_0 ∨ **John**_1 ∨ **Mary**_2} ∨ {**loves**_0 ∨ **Tom**_1 ∨ **Wendy**_2}]. (8.7)

Let us note that this example may be represented using the role-filler scheme of Plate's HRRs as

L1 = loves + lover∗**Tom + loved**∗**Wendy**, (8.8)
L2 = loves + lover∗**John + loved**∗**Mary**, (8.9)
L = L1 + L2. (8.10)

Under such a representation, the information about who loves whom is lost in L (Plate, 1995; Halford, Wilson, & Phillips, in press). In our representation, this information is preserved even using the role-filler scheme:

L1 = [**loves** ∨ {**lover** ∨ **Tom**} ∨ {**loved** ∨ **Wendy**}], (8.11)
L2 = [**loves** ∨ {**lover** ∨ **John**} ∨ {**loved** ∨ **Mary**}], (8.12)
L = <**L1** ∨ **L2**>. (8.13)

Another example of relational instance from Halford, Wilson, & Phillips (in press):

cause(shout-at(John,Tom),hit(Tom,John)). (8.14)

Using our representation scheme, it may be represented as

[**cause** ∨ {**shout-at**_0 ∨ **John**_1 ∨ **Tom**_2}_1 ∨ {**hit**_0 ∨ **Tom**_1 ∨ **John**_2}_2]. (8.15)

8.3.3. Tree-like structure

An example of bracketed binary tree from Pollack (1990):

((D (A N)) (V (P (D N)))). (8.16)

If we do not take the order into account, but use only the information about the grouping of constituents, our representation may look as simple as:

/<**D** ∨ [**A** ∨ **N**]> ∨ <**V** ∨ [**P** ∨ {**D** ∨ **N**}]>/. (8.17)

8.3.4 Labeled directed acyclic graph

Sperduti & Starita (1997), Frasconi, Gori, & Sperduti (1997) provide examples of labeled directed acyclic graphs. Let us consider

F(a, f(y), f(y,F(a,b))). (8.18)

Using our representation, its representation may look as

<**F**_0 ∨ **a**_1 ∨ [**f**_0 ∨ **y**_1]_2 ∨ [**f**_0 ∨ **y**_1 ∨ {**F**_0 ∨ **a**_1 ∨ **b**_2}_2]_3 >. (8.19)

9. Related work and discussion

The procedures of Context-Dependent Thinning allow to implement binary sparse representations of complex data structures, including nested compositional structures or part-whole hierarchies. The basic principles of such representations and their use for data handling were proposed in the frames of the Associative-Projective Neural Networks paradigm (Kussul, 1988, 1992; Kussul, Rachkovskij, & Baidyk, 1991a).

9.1 Comparison to other representation schemes

Let us compare our scheme for representation of complex data structures using the CDT procedure (we will call it "APNN-CDT" below) with other schemes using distributed representations. The best known schemes are (L)RAAMs (Pollack, 1990; Blair, 1997; Sperduti 1994), Tensor Product representations (Smolensky, 1990; Halford, Wilson, & Phillips, in press), Holographic Reduced Representations (HRRs) (Plate, 1991, 1995), Binary Spatter Codes (Kanerva, 1996). For this comparison, we will use the framework of Plate (1997) who proposes to distinguish these schemes using the following features: the nature of distributed representation; the choice of superposition; the choice of binding operation; how the binding operation is used to represent predicate structure; the use of other operations and techniques.

9.1.1. The nature of distributed representation

Vectors of random real-valued elements with the Gaussian distribution are used in HRRs. Dense binary random codes with the number of 1s equal to the number of 0s are used in Binary Spatter Codes. Vectors with real or binary elements (without specified distributions) are used in other schemes.

In the APNN-CDT scheme, binary vectors with randomly distributed small number of 1s are used to encode base-level items.

9.1.2. The choice of superposition

The operation of superposition is used for unstructured representation of an aggregate of codevectors. It is such kind of representation that caused the criticism of distributed schemes for the lack of structure.

In the Binary Spatter Codes superposition is realized as a bitwise thresholded addition of codevectors. Schemes with non-binary elements use elementwise summation. In HRRs, it is

performed in the explicit form. In (L)RAAMs, the input summation of hidden units realizes superposition. For tensors, superposition is realized as adding up or ORing the corresponding elements.

In the APNN-CDT scheme elementwise OR is used.

9.1.3. The choice of binding operation

Most schemes use special two-place operation for binding of codevectors. The resulting bound vector has the same dimension as initial codevectors (or one of them in (L)RAAMs). The binding operation is performed "on the fly" by circular convolution (HRRs), elementwise multiplication (Gayler, 1998), or XOR (Binary Spatter Codes). In (L)RAAMs, binding is realized through multiplication of an input codevector by the weight matrix of the hidden layer formed by training of a multilayer perceptron.

The vector obtained by binding can be bound with another codevector in its turn. In Tensor Models, binding of several codevectors is performed by their tensor product. The dimensionality of resulting tensor grows with the number of bound codevectors.

In the APNN-CDT scheme, binding is performed by the Context Dependent Thinning procedure. Unlike the other schemes where the codevectors to be bound are initially separated, they can be superimposed by disjunction in the basic version of the CDT procedure. Superposition codevector **Z** (as in equation 4.4) makes the context codevector. The result of the CDT procedure may be considered as superimposed bindings of each constituent codevector with the context codevector. Or, it may be considered as superimposed paired bindings of all constituent codevectors with each other. (Note that in the “selfless” CDT version (section 5) the codevector of each constituent is not bound to itself. In the hetero-CDT version, one codevector is bound to another codevector through thinning from the latter).

According to Plate's framework, CDT as a binding procedure can be considered as a special kind of superposition (disjunction) of certain elements of the tensor product of **Z** by itself (i.e. N^2 scalar products Z_iZ_j). Actually, $\{Z_i\}$ is disjunction of certain $Z_iZ_j=Z_i \wedge Z_j$, where Z_j is a permutated element of Z_i (equation 4.10). CDT can be also considered as a hashing procedure: the subspace to where hashing is performed is defined by 1s of **Z**, and some 1s of **Z** are mapped to that subspace.

Since the resulting bound codevector $\{\mathbf{Z}\}$ is obtained in the CDT procedure by thinning of 1s of vector **Z** (in which constituent codevectors are superimposed), $\{\mathbf{Z}\}$ is similar to all constituent codevectors (unstructured similarity is preserved). Therefore to retrieve the composition of constituents bound in the thinned codevector, we only need to choose the most similar items from the alphabet of constituent codevectors. This can be done in parallel using an associative memory.

All other known binding operations do not preserve unstructured similarity. Therefore to extract some constituent codevector from the bound codevector, they demand to know the other constituent codevector(s). Then rebinding of the bound codevector with the inverses of known constituent codevector(s) produces a noised version of the sought constituent codevector. (For some representation schemes a codevector is its own inverse). This operation is known as "unbinding". To eliminate noise from the unbound codevector, a "clean-up" memory with the full alphabet of constituent codevectors is also required in those schemes. If some or all constituents of the bound codevector are not known, decoding in those schemes requires exhaustive search (substitution, binding, and checking) through all combinations of

codevectors from the alphabet. Then the obtained bound codevector most similar to the bound codevector to be decode provides the information on its composition.

As in the other schemes, structured similarity is preserved by the CDT, i.e. bindings of similar patterns are similar to each other. However the character of similarity is different. In most of the other schemes the similarity of the bound codevectors is equal to the product of similarities of the component codevectors (e.g. Plate, 1995). For example, the similarity of **A**∗**B** and **A**∗**B'** is equal to the similarity of **B** and **B'**. Therefore if **B** and **B'** are not similar at all, the bound vectors also will not be similar.

The codevectors to be bound by the CDT procedure are initially superimposed component codevectors. So their initial similarity is the mean of the components' similarities. Also, the thinning itself preserves approximately the square of similarity of the input vectors. So, the similarity for our example will be >0.25 instead of 0 for the other schemes.

9.1.4. How the binding operation is used to represent predicate structure

In most of the schemes predicate structures are represented by role-filler bindings. Halford, Wilson, & Phillips (in press) use predicate-argument bindings. The APNN-CDT scheme allows such representations of predicate structures as role-filler bindings, predicate-argument bindings, and also offers a potential for other possible representations. Both ordered and unordered arguments can be represented.

9.1.5. The use of other operations and techniques.

- *Normalization.* After superposition of codevectors, some normalizing transformation is used in various schemes to support the individual elements or the total strength of the resulting codevector within certain limits. In Binary Spatter Codes, it is the threshold operation that converts a non-binary codevector (the bitwise sum of component codevectors) to a binary one. In (L)RAAMs, it is the sigmoid non-linearity of hidden units through which the input activation (the sum of role-filler bindings) is passed. In HRRs, it is the scaling of codevectors to the unit length that facilitates their comparison.

The CDT procedure performs a dual role: not only it binds superimposed codevectors of constituents, but it also normalizes the density of the resulting codevector. It would be interesting to check if normalization operations in other schemes provide the effect of binding as well.

Clean-up memory. Associative memory is used in various representation schemes for storage of constituent codevectors and their recall (clean-up after finding their approximate noisy versions using unbinding). After the CDT procedure, the combined codevector is similar to the constituent codevectors, however the latter are represented in the reduced form. Therefore it is naturally to use associative memories in the APNN-CDT scheme to store and retrieve the codevectors of component items of various complexity levels. Since the constituent codevectors of different complexity levels have approximately the same and small number of 1s, an associative memory based on assembly neural networks with simple Hebbian learning rule allows efficient manipulation with a large number of codevectors.

Chunking. The problem of chunking remains one of the least developed problems in existing representation schemes.

In the HRRs and the Binary Spatter Codes chunks are normalized superpositions of stand-alone constituent codevectors and their bindings. In its turn, the codevector of a chunk can be used as one of the components for binding. Thus, chunking allows to build structures of arbitrary nesting or composition level. Each chunk should be stored in a clean-up memory. When complex structures are decoded by unbinding, noisy versions of chunk codevectors are obtained. They are used to retrieve pure versions from the clean-up memory, which can be decoded in their turn.

In those schemes, the codevectors of chunks are not bound. Therefore they can not be superimposed without the risk of structure loss, as it was repeatedly mentioned in this paper. In the APNN-CDT scheme, any combined codevector after thinning represents a chunk. Since the component codevectors are bound in the chunk codevector, it can be operated as a single whole (an entity) without confusion of components belonging to different items.

When a compositional structure is constructed using HRRs or Binary Spatter Codes, the chunk codevector is usually the filler which becomes bound with some role codevector. In this case, in distinction to the APNN-CDT scheme, the components **A**,**B**,**C** of the chunk become bound with the role rather than with each other:

$$\mathbf{role}*(\mathbf{A} + \mathbf{B} + \mathbf{C}) = \mathbf{role}*\mathbf{A} + \mathbf{role}*\mathbf{B} + \mathbf{role}*\mathbf{C}. \quad (9.1)$$

Again, if the role in not unique, it can not be determined to which chunk the binding **role**∗**A** belongs. Also, the role codevector should be known for unbinding and subsequent retrieving of the chunk.

Thus in the representation schemes of HRRs and Binary Spatter Codes each of the component codevectors belonging to a chunk binds with (role) codevectors of other hierarchical levels not belonging to that chunk. Therefore such bindings may be considered as "vertical". In the APNN-CDT scheme, a "horizontal" binding is essential: the codevectors of the chunk components are bound with each other.

Gayler (1998) proposes to bind a chunk codevector with its permutated version. It resembles the version of thinning procedure from section 4.3, but for real-valued codevectors. Different codevector permutations for different nesting levels allow to distinguish the components of chunks from different levels, in a similar fashion as different configurations of thinning connections in the CDT. Without it, the vertical binding in the schemes of Plate, Kanerva, and Gayler, such as **role_upper_level** ∗ (**role_lower_level** ∗ **filler**) is indistinguishable from **role_lower_level** ∗ (**role_upper_level** ∗ **filler**), because their binding operations are associative and commutative. For the CDT procedure, in contrast , $[\{\mathbf{A} \vee \mathbf{B}\} \vee \mathbf{C}] \neq [\mathbf{A} \vee \{\mathbf{B} \vee \mathbf{C}\}]$.

However since the result of binding in the scheme of Gayler and in the other considered schemes (with the exception of APNN-CDT) is not similar to the component items, in those schemes decoding of the chunk codevector created by binding with a permutation of itself will generally require exhaustion of all combinations of component codevectors.

The obvious problem of Tensor Product representation is to avoid the growth of dimensionality of the resulting pattern obtained by the binding of components. Otherwise the dimensionality will grow exponentially with the nesting depth. Halford, Wilson, & Phillips (in press) consider chunking as the means to reduce the rank of tensor representation. To realize chunking, they propose to use the operations of convolution, concatenation, superposition, as well as some special function that associates the outer product with the codevector of lower

dimension. However the first three operations do not rule out confusion of grouping or ordering of arguments inside chunk, (i.e., different composite items may produce identical chunks). And the special function (and its inverse) requires concrete definition. Probably it could be done using associative memory, e.g. of the sigma-pi type proposed by Plate (1998).

In (L)RAAMs the chunks of different nesting levels are encoded in the same weight matrix of connections between the input layer and the hidden layer of multilayer perceptron. It may be one of the reasons for poor generalization. Probably if additional multilayer perceptrons are introduced for each nesting level (with the input for each following perceptron provided by the hidden layer of the preceding one, similarly to Sperduti & Starita, 1997), generalization in those schemes would improve.

In the APNN, chunks (combined codevectors) of different nesting levels are memorized in different autoassociative neural networks.

9.2. Sparse binary schemes

Indicating unknown areas where useful representation schemes for nested compositional structures can be found, Plate (1997) notes that known schemes poorly handle sparse binary patterns, because known binding and superposition operations change the density of sparse patterns.

From the work known to us, only Sjodin (1998) expresses the idea of "thinning" in an effort to avoid the low associative memory capacity for dense binary patterns. He defines the thinning operation as preserving the 1s corresponding to the maximums of some function defined over the binary vector. The values of that function are determined at cyclic shifts of the codevector by the number of steps equal to the ordering number of 1s in that codevector. However it is not clear from such a description what are the properties of the maximums, and, therefore, whether the similarity requirements of section 3.4 are satisfied and what is the character of similarity.

The CDT procedure considered in this paper allows to preserve the density of codevectors while binding them. Coupled with the techniques for encoding of the pattern ordering, this procedure allows to implement various representation schemes of complex structured data. Approximately the same low density of binary codevectors at different nesting levels permits to use identical procedures of construction, recognition, comparison, and decoding of patterns at different hierarchical levels of the Associative-Projective Neural Network architecture (Kussul 1988, 1992; Kussul, Rachkovskij, & Baidyk, 1991).

The CDT procedure preserves the similarity of encoded descriptions allowing to determine the similarity of complex structures by the overlap of their codevectors. Also, in the codevectors of complex structures formed using the CDT procedure, representation of the component codevectors (their 1s) is reduced. Therefore, the APNN-CDT scheme may pretend to be another implementation of Hinton's (1990) reduced descriptions, Amosov's (1968) item coarsing, or compression of Halford, Wilson, & Phillips (in press). Besides, the CDT scheme is biologically relevant since it uses sparse representations and allows simple neural network implementation.

10. Conclusion

Procedures of Context-Dependent Thinning described in the paper perform binding of items represented as sparse binary codevectors. They allow to bind on the fly a variable number of superimposed patterns while preserving the density of combined codevectors. The result of the CDT is of the same dimensionality as the constituent codevectors. Using of the auto-CDT procedures as analogs of brackets in the bracketed symbolic representation of various complex data structures permits easy transformation of these representations to the binary codevectors of fixed dimensionality with a small number of 1s.

Unlike other binding procedures, binding by the auto-CDT preserves the similarity of combined codevector with each of the constituent codevectors. It allows both to determine the similarity of complex items with each other by the overlap of their codevectors and to retrieve in full size the codevectors of their constituents. Such operations are efficiently implementable by distributed associative memories which provide high information capacity for the codevectors with small number of 1s.

The APNN-CDT style representations have been already used by us in applications (an earlier work is reviewed in Kussul, 1992, 1993; more recent developments are described in Lavrenyuk, 1995; Rachkovskij, 1996; Kussul & Kasatkina, 1999). We hope that the CDT procedures will find their application for distributed representation and manipulation of complex compositional data structures, contributing to the progress of connectionist symbol processing (Touretzky, 1995, 1990; Touretzky & Hinton, 1988; Hinton, 1990; Plate, 1995). Fast (parallel) evaluation of similarity or finding the most similar structure allowed by such a representation are extremely useful for solution of a wide range of AI problems (Kussul 1992,1993; Hinton, 1990; Plate, 1998).

Acknowledgements: This work was supported in part by the International Science Foundation Grants U4M000 and U4M200, as well as by the INTAS Project 93-0560.

References

Amari, S. (1989). Characteristics of sparsely encoded associative memory. *Neural Networks, 2*, 445-457.

Amosov, N. M. (1967). *Modeling of thinking and the mind*. New York: Spartan Books.

Amosov, N. M., Baidyk, T. N., Goltsev, A. D., Kasatkin, A. M., Kasatkina, L. M., Kussul, E. M., & Rachkovskij, D. A. (1991). *Neurocomputers and intelligent robots*. Kiev: Naukova dumka. (In Russian).

Artykutsa, S. Ya., Baidyk, T. N., Kussul, E. M., & Rachkovskij, D. A. (1991). Texture recognition using neurocomputer. (Preprint 91-8). Kiev, Ukraine: V. M. Glushkov Institute of Cybernetics. (In Russian).

Baidyk, T. N., Kussul, E. M., & Rachkovskij, D. A. (1990). Numerical-analytical method for neural network investigation. In *Proceedings of The International Symposium on Neural Networks and Neural Computing - NEURONET'90* (pp. 217-222). Prague, Czechoslovakia.

Blair, A. D. (1997). Scaling-up RAAMs. (Technical Report CS-97-192). Brandeis University, Department of Computer Science.

Fedoseyeva, T. V. (1992). The problem of training neural network training to recognize word roots. In *Neuron-like networks and neurocomputers* (pp. 48-54). Kiev, Ukraine: V. M. Glushkov Institute of Cybernetics. (In Russian).

Feldman, J. A., & Ballard, D. H. (1982). Connectionist models and their properties. *Cognitive Science, 6*, 205-254.

Feldman, J. A. (1989). Neural Representation of Conceptual Knowledge. In L. Nadel, L. A. Cooper, P. Culicover, & R. M. Harnish (Eds.), *Neural connections, mental computation* (pp. 68-103). Cambridge, Massachusetts, London, England: A Bradford Book, The MIT Press.

Foldiak, P., & Young, M. P. (1995). Sparse Coding in the Primate Cortex. In M. A. Arbib (Ed.), *Handbook of brain theory and neural networks* (pp. 895-898). Cambridge, MA: MIT Press.

Frasconi, P., Gori, M., & Sperduti, A. (1997). A general framework for adaptive processing of data structures. *Technical Report DSI-RT-15/97.* Firenze, Italy: Universita degli Studi di Firenze, Dipartimento di Sistemi e Informatica.

Frolov, A. A., & Muraviev, I. P. (1987). *Neural models of associative memory*. Moscow: Nauka. (In Russian).

Frolov, A. A., & Muraviev, I. P. (1988). Informational characteristics of neuronal and synaptic plasticity. *Biophysics, 33*, 708-715.

Frolov, A. A. (1989). Information properties of bilayer neuron nets with binary plastic synapses. *Biophysics, 34*, 868-876.

Gayler, R. W. (1998). Multiplicative binding, representation operators, and analogy. In K. Holyoak, D. Gentner, & B. Kokinov (Eds.), *Advances in analogy research: Integration of theory and data from the cognitive, computational, and neural sciences*. (p. 405). Sofia, Bulgaria: New Bulgarian University. (Poster abstract. Full poster available at: http://cogprints.soton.ac.uk/abs/comp/199807020).

Halford, G. S., Wilson W. H., & Phillips S. (in press). Processing capacity defined by relational complexity: implications for comparative, developmental, and cognitive psychology. *Behavioral and Brain Sciences.*

Hebb, D. O. (1949). *The organization of behavior*. New York: Wiley.

Hinton, G. E. (1981). Implementing semantic networks in parallel hardware. In G. E. Hinton & J. A. Anderson (Eds.), *Parallel models of associative memory* (pp. 161-187). Hillside, NJ: Lawrence Erlbaum Associates.

Hinton, G. E. (1990). Mapping part-whole hierarchies into connectionist networks. *Artificial Intelligence, 46*, 47-76.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences, USA, 79*, 2554-2558.

Hopfield, J. J., Feinstein, D. I., & Palmer, R. G. (1983). "Unlearning" has a stabilizing effect in collective memories. *Nature, 304*, 158-159.

Hummel, J. E., & Holyoak K. J. (1997). Distributed representations of structure: A theory of analogical access and mapping. *Psychological Review, 104*, 427-466.

Kanerva, P. (1996). Binary Spatter-Coding of Ordered K-tuples. In C. von der Malsburg, W. von Seelen, J. C. Vorbruggen, & B. Sendhoff (Eds.). *Proceedings of the International Conference on Artificial Neural Networks - ICANN'96, Bochum, Germany. Lecture Notes in Computer Science, 1112,* 869-873. Berlin: Springer.

Kasatkin, A. M., & Kasatkina, L. M. (1991). A neural network expert system. In *Neuron-like networks and neurocomputers* (pp. 18-24). Kiev, Ukraine: V. M. Glushkov Institute of Cybernetics. (In Russian).

Kussul, E. M. (1980). *Tools and techniques for development of neuron-like networks for robot control*. Unpublished Dr. Sci. dissertation. Kiev, Ukrainian SSR: V. M. Glushkov Institute of Cybernetics. (In Russian).

Kussul, E. M. (1988). Elements of stochastic neuron-like network theory. In *Internal Report "Kareta-UN"* (pp. 10-95). Kiev, Ukraine: V. M. Glushkov Institute of Cybernetics. (In Russian).

Kussul, E. M. (1992) *Associative neuron-like structures*. Kiev: Naukova Dumka. (In Russian).

Kussul, E. M. (1993). On some results and prospects of development of associative-projective neurocomputers. In *Neuron-like networks and neurocomputers* (pp. 4-11). Kiev, Ukraine: V. M. Glushkov Institute of Cybernetics. (In Russian).

Kussul, E. M., & Baidyk, T. N. (1990). Design of a neural-like network architecture for recognition of object shapes in images. *Soviet Journal of Automation and Information Sciences, 23(5),* 53-58.

Kussul, E. M., & Baidyk, T. N. (1993). On information encoding in associative-projective neural networks. (Preprint 93-3). Kiev, Ukraine: V. M. Glushkov Institute of Cybernetics. (In Russian).

Kussul, E. M., Baidyk, T. N., Lukovich, V. V., & Rachkovskij, D. A. (1993). Adaptive neural network classifier with multifloat input coding. In *Proceedings of NeuroNimes'93, Nimes, France, Oct. 25-29, 1993*. EC2-publishing.

Kussul, E. M., & Kasatkina, L. M. (1999). Neural network system for continuous handwritten words recognition. (Submitted).

Kussul, E. M., & Rachkovskij, D. A. (1991). Multilevel assembly neural architecture and processing of sequences. In A .V. Holden & V. I. Kryukov (Eds.), *Neurocomputers and Attention: Vol. II. Connectionism and neurocomputers* (pp. 577-590). Manchester and New York: Manchester University Press.

Kussul, E. M., Rachkovskij, D. A., & Baidyk, T. N. (1991a). Associative-Projective Neural Networks: architecture, implementation, applications. In *Proceedings of the Fourth International Conference "Neural Networks & their Applications", Nimes, France, Nov. 4-8, 1991* (pp. 463-476).

Kussul, E. M., Rachkovskij, D. A., & Baidyk, T. N. (1991b). On image texture recognition by associative-projective neurocomputer. In C. H. Dagli, S. Kumara, & Y. C. Shin (Eds.), *Proceedings of the ANNIE'91 conference "Intelligent engineering systems through artificial neural networks"* (pp. 453-458). ASME Press.

Lansner, A., & Ekeberg, O. (1985). Reliability and speed of recall in an associative network. *IEEE Trans. Pattern Analysis and Machine Intelligence, 7*, 490-498.

Lavrenyuk, A. N. (1995). Application of neural networks for recognition of handwriting in drawings. In *Neurocomputing: issues of theory and practice* (pp. 24-31). Kiev, Ukraine: V. M. Glushkov Institute of Cybernetics. (In Russian).

Legendy, C. R. (1970). The brain and its information trapping device. In Rose J. (Ed.), *Progress in cybernetics, vol. I*. New York: Gordon and Breach.

Milner, P. M. (1974). A model for visual shape recognition. *Psychological Review, 81*, 521-535.

Milner, P.M. (1996). Neural representations: some old problems revisited. *Journal of Cognitive Neuroscience, 8*, 69-77.

Palm, G. (1980). On associative memory. *Biological Cybernetics, 36*, 19-31.

Palm, G. (1993). The PAN system and the WINA project. In P. Spies (Ed.), *Euro-Arch'93* (pp. 142-156). Springer-Verlag

Palm, G. & Bonhoeffer, T. (1984). Parallel processing for associative and neuronal networks. *Biological Cybernetics, 51*, 201-204.

Plate, T. A. (1991). Holographic Reduced Representations: Convolution algebra for compositional distributed representations. In J. Mylopoulos & R. Reiter (Eds.), *Proceedings of the 12th International Joint Conference on Artificial Intelligence* (pp. 30-35). San Mateo, CA: Morgan Kaufmann.

Plate, T. A. (1995). Holographic Reduced Representations. *IEEE Transactions on Neural Networks, 6*, 623-641.

Plate, T. (1997). A common framework for distributed representation schemes for compositional structure. In F. Maire, R. Hayward, & J. Diederich (Eds.), *Connectionist systems for knowledge representation and deduction* (pp. 15-34). Queensland University of Technology.

Plate, T. (1998). Analogy retrieval and processing with distributed representations. In K. Holyoak, D. Gentner, & B. Kokinov (Eds.), *Advances in analogy research: Integration of theory and data from the cognitive, computational, and neural sciences* (pp. 154-163). Sofia, Bulgaria: New Bulgarian University.

Plate, T. (1998). Analogy retrieval and processing with distributed represenations. *Technical report CS-TR-98-4*. Wellington, New Zealand: Victoria University of Wellington, School of Mathematical and Computer Science.

Plate, T. (submitted). Randomly connected sigma-pi neurons can form associative memories.

Pollack, J. B. (1990). Recursive distributed representations. *Artificial Intelligence, 46*, 77-105.

Rachkovskij, D. A. (1990a). On numerical-analytical investigation of neural network characteristics. In *Neuron-like networks and neurocomputers* (pp. 13-23). Kiev, Ukraine: V. M. Glushkov Institute of Cybernetics. (In Russian).

Rachkovskij, D. A. (1990b). *Development and investigation of multilevel assembly neural networks*. Unpublished Ph.D. dissertation. Kiev, Ukrainian SSR: V. M. Glushkov Institute of Cybernetics. (In Russian).

Rachkovskij, D. A. (1996). Application of stochastic assembly neural networks in the problem of interesting text selection. In *Neural network systems for information processing (*pp. 52-64). Kiev, Ukraine: V. M. Glushkov Institute of Cybernetics. (In Russian).

Rachkovskij, D. A. & Fedoseyeva, T. V. (1990). On audio signals recognition by multilevel neural network. In *Proceedings of The International Symposium on Neural Networks and Neural Computing - NEURONET'90 (*pp. 281-283). Prague, Czechoslovakia.

Rachkovskij, D. A. & Fedoseyeva T. V. (1991). Hardware and software neurocomputer system for recognition of acoustical signals. In *Neuron-like networks and neurocomputers* (pp. 62-68). Kiev, Ukraine: V. M. Glushkov Institute of Cybernetics. (In Russian).

Shastri, L. & Ajjanagadde, V. (1993). From simple associations to systematic reasoning: connectionist representation of rules, variables, and dynamic bindings using temporal synchrony. *Behavioral and Brain Sciences, 16*, 417-494.

Sjodin, G. (1998). The Sparchunk Code: a method to build higher-level structures in a sparsely encoded SDM. In *Proceedings of IJCNN'98* (pp.), IEEE, Piscataway, NJ: IEEE.

Sjodin, G., Kanerva, P., Bjorn, L., & Kristoferson, J. (1998). Holistic higher-level structure-forming algorithms. In *Proceedings of 1998 Real World Computing Symposium - RWC'98* (pp. 299-304).

Smolensky, P. (1990). Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial Intelligence, 46*, 159-216.

Sperduti, A. (1994). Labeling RAAM. *Connection Science, 6*, 429-459.
Sperduti, A. & Starita, A. (1997). supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks, 8*, 714-735.
Tsodyks, M. V. (1989). Associative memory in neural networks with the Hebbian learning rule. *Modern Physics Letters B, 3*, 555-560.
Touretzky, D. S. (1990). BoltzCONS: Dynamic symbol structures in a connectionist network. *Artificial Intelligence, 46*, 5-46.
Touretzky, D. S. (1995). Connectionist and symbolic representations. In M. A. Arbib (Ed.), *Handbook of brain theory and neural networks* (pp. 243-247). Cambridge, MA: MIT Press.
Touretzky, D. S., & Hinton, G. E. (1988). A distributed connectionist production system. *Cognitive Science, 12*, 423-466.
Vedenov, A. A. (1987). "False memory" in model neural networks. (Preprint IAE-4395/1). Moscow: I. V. Kurchatov Institute of Atomic Energy.
Vedenov, A. A. (1988). *Modeling of thinking elements*. Moscow: Science. (In Russian).
von der Malsburg, C. (1981). The correlation theory of brain function. (Internal Report 81-2). Gottingen, Germany: Max-Planck-Institute for Biophysical Chemistry, Department of Neurobiology.
von der Malsburg, C. (1985). Nervous structures with dynamical links. *Ber. Bunsenges. Phys. Chem., 89*, 703-710.
von der Malsburg, C. (1986) Am I thinking assemblies? In G. Palm & A. Aertsen (Eds.), *Proceedings of the 1984 Trieste Meeting on Brain Theory* (pp. 161-176). Heidelberg: Springer-Verlag.
Willshaw, D. J., Buneman, O. P., & Longuet-Higgins, H. C. (1969). Non-holographic associative memory. *Nature, 222*, 960-962.

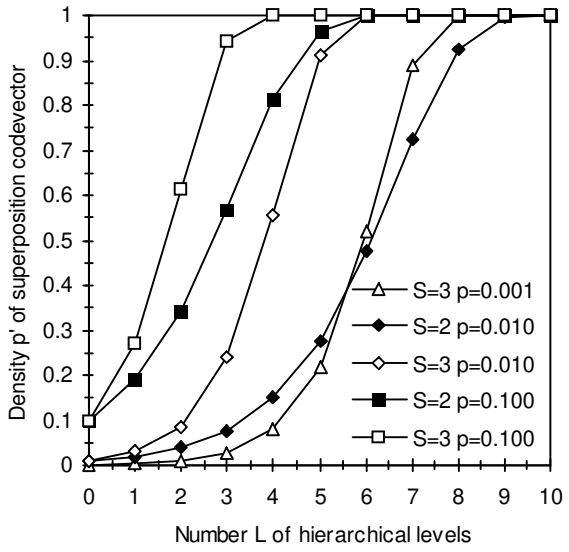


Figure 1. Growth of the density p' of 1s in the composite codevectors of higher hierarchical levels (see equations 2.1 and 2.2). Here each codevector of the higher level is formed by bit disjunction of S codevectors of the preceding level. Items of all levels are considered uncorrelated. The codevectors of base-level items are independent with the density of 1s equal to p. The number of hierarchical levels is L. (For fixed number of base-level items, the total number of 1s in the composite codevectors is obviously limited by the number of 1s in the disjunction of base-level codevectors).

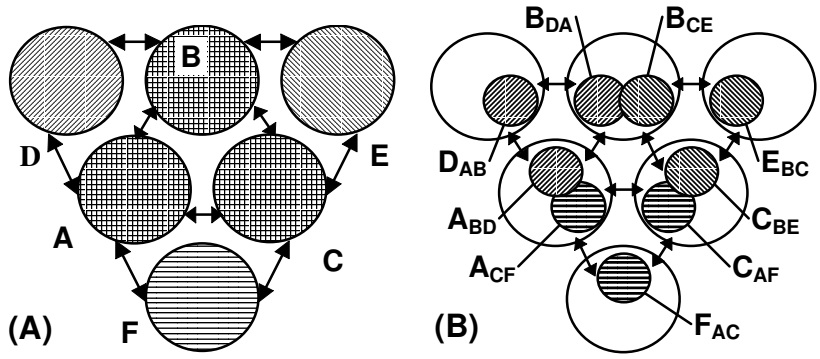


Figure 2. (A) Formation of a "false" assembly. When three assemblies (ABD; BCE; ACF) are consecutively formed in a neural network by connecting all active units of patterns encoding their items, the fourth assembly ABC is formed as well, though its pattern was not explicitly presented to the network. (B) Preventing of a "false" assembly. If each of three assemblies is formed by connecting only subsets of active units encoding the component items, then the connectivity of the false assembly is weak. Here X_{YZ} denotes the subset of units encoding item X when items Y and Z are also present. Shaded circles represent patterns of active units encoding items; formed connections are plotted by arrowed lines.

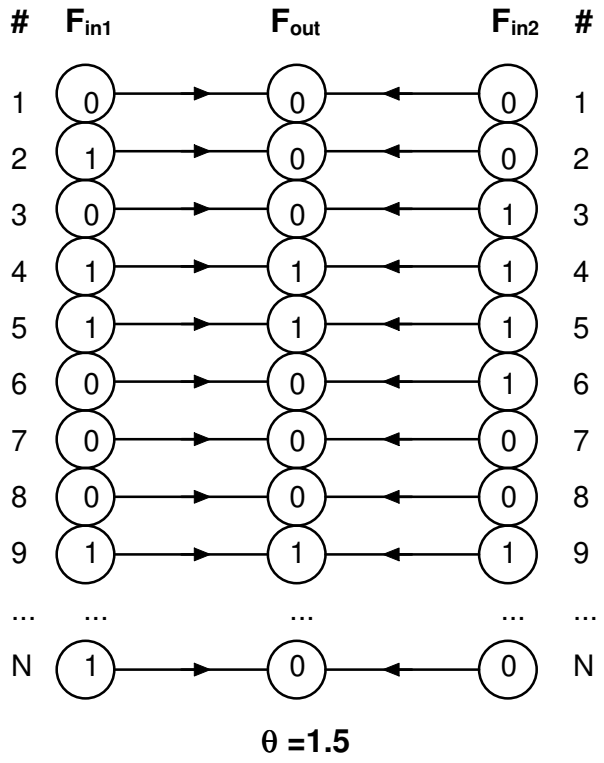


Figure 3. A neural network implementation of direct conjunctive thinning of two codevectors. Two N-dimensional binary patterns corresponding to the codevectors are activated in the input neural fields F_{in1} and F_{in2}. The result of their elementwise conjunction is in the neural field F_{out}, where the neural threshold θ=1.5.

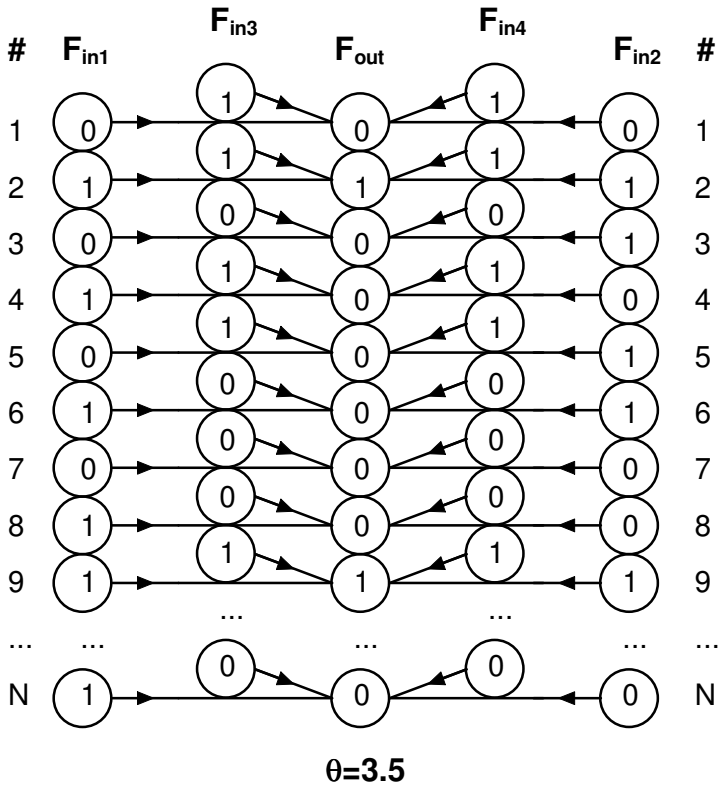


Figure 4. A neural network implementation of direct conjunctive thinning of several codevectors. An example with four codevectors is shown (S=4). Each corresponding N-dimensional binary pattern is activated in one of the input neural fields F_{in1}, F_{in2}, F_{in3}, F_{in4}. For this case the threshold of the neurons of the output field θ=3.5, so that the network performs bit conjunction of four input codevectors.

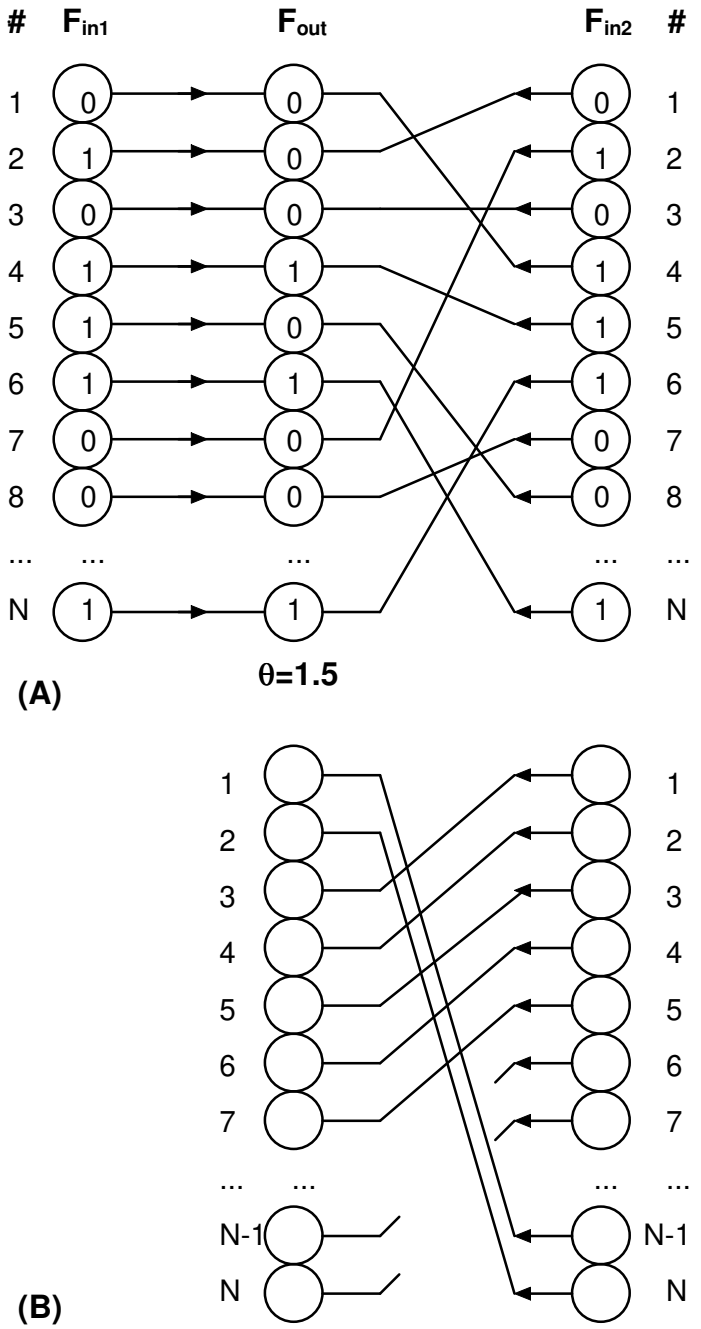


Figure 5. (A) A neural network implementation of paired conjunctive thinning. The same N-dimensional binary patterns are input in F_{in1} and F_{in2}. They are superimposed codevectors of several component items. F_{in1} is connected with F_{out} by a bundle of direct projective connections. F_{in2} is connected with F_{out} by a bundle of permutative connections. Conjunction of the superimposed component codevectors and their permutation is obtained in F_{out}. (B) Implementation of a bundle of permutative connections by "shift" connections. Here the shift is 2.

The input codevector $\mathbf{Z} = \mathbf{X}_1 \vee \mathbf{X}_2 \vee ... \vee \mathbf{X}_S$ **(A)**
The output (thinned) codevector is {**Z**}
Calculate K=ln(1-1/S)/ln(1-pS)
Set auxiliary vector **W** to 0
Seed the random numbers' generator: randomize(*seed*).
Note: *seed* defines the unique configuration of shift permutations

```
for(k=0; k<K; k=k+1)
        if ((r=rand()) ≠ 0)
                for(i=0; i<N; i=i+1)
                        W_i = W_i ∨ Z_{i+r modulo N};
for(i=0; i<N; i=i+1)
        {Z}_i = Z_i ∧ W_i
```

The input codevector $\mathbf{Z} = \mathbf{X}_1 \vee \mathbf{X}_2 \vee ... \vee \mathbf{X}_S$ **(B)**
Set the output (thinned) codevector {**Z**} to 0.
Calculate K=ln(1-1/S)/ln(1-pS)
Set an auxiliary vector **W** to 0
Seed the random numbers' generator: randomize(*seed*).
Note: *seed* defines a unique configuration of shift permutations

```
while(|{Z}| < M)
        if ((r=rand()) ≠ 0)
                for(i=1; i≤N; i=i+1)
                        {Z}_i = {Z}_i ∨ (Z_i ∧ Z_{i+r modulo N});
```

Figure 6. Algorithmic implementations of the additive version of Context Dependent Thinning procedure. (A) The first example. (B) The second example

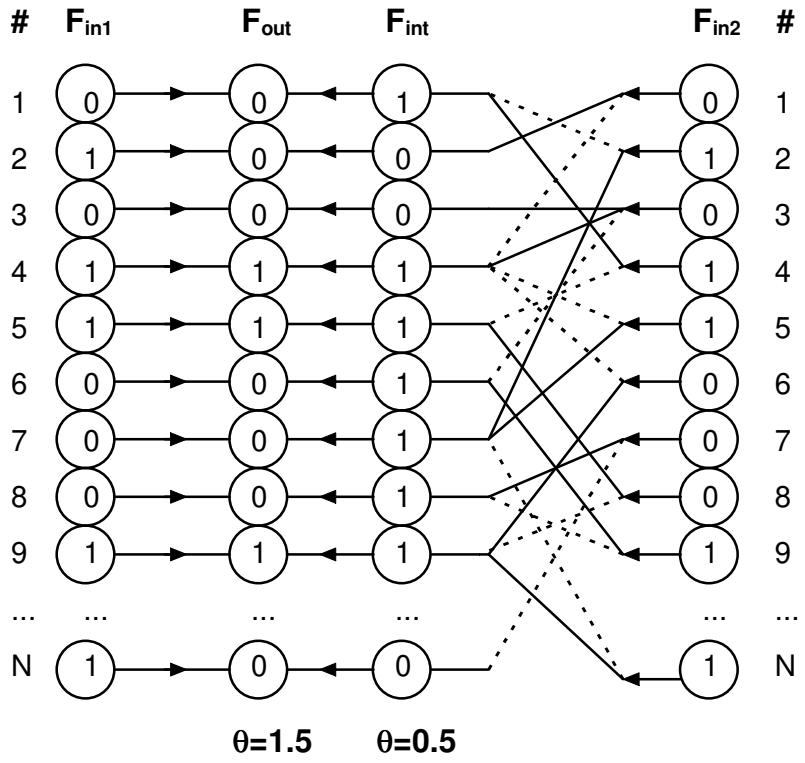


Figure 7. A neural network implementation of the additive version of Context-Dependent Thinning procedure. The same patterns corresponding to superimposed codevectors of component items are in the input fields F_{in1} and F_{in2}. The output field F_{out} executes conjunction of patterns in F_{in1} and F_{int}. The intermediate field F_{int} is connected to the input field F_{in2} by a number K of bundles of permutative connections. Only two bundles are shown here: one by solid lines and one by dotted lines. F_{int} accumulates various permutations of the pattern in F_{in2} until its conjunction with the pattern in F_{in1} provides the required density of the output pattern in F_{out}. The number K of required bundles is estimated in Table 2. A mechanism not shown in the figure provides "switching on" the permutative bundles in fixed order and estimates the density of patterns in the fields.

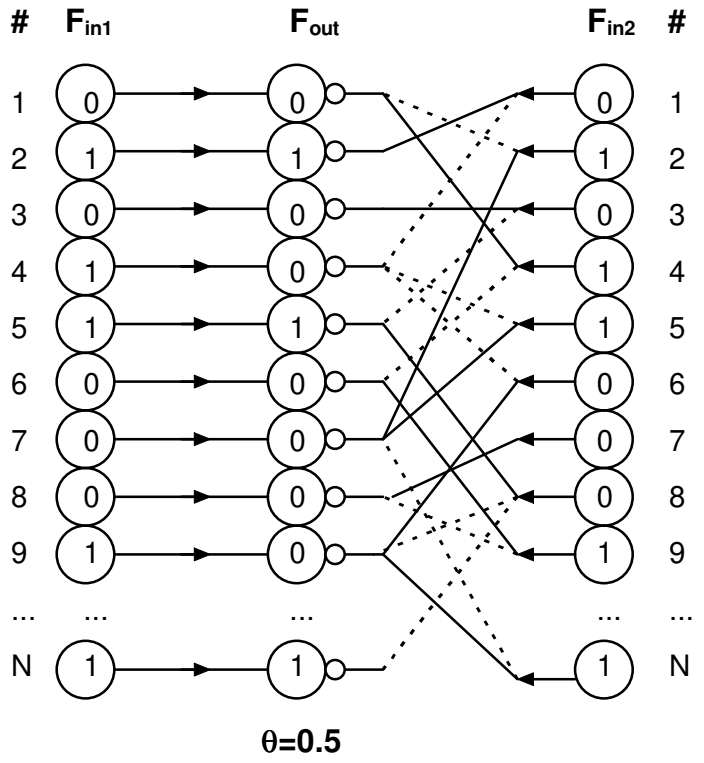
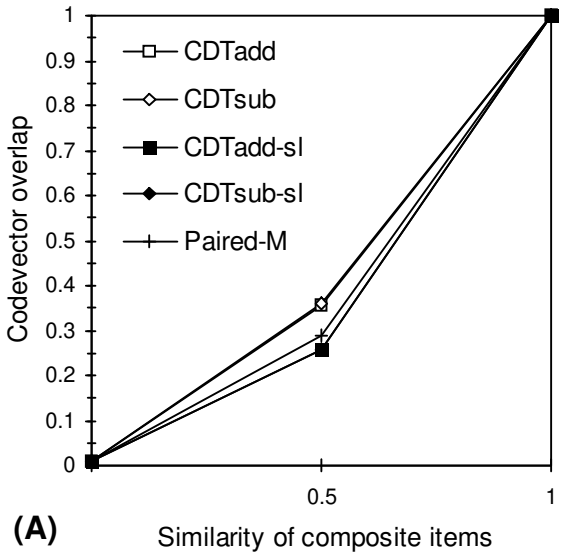
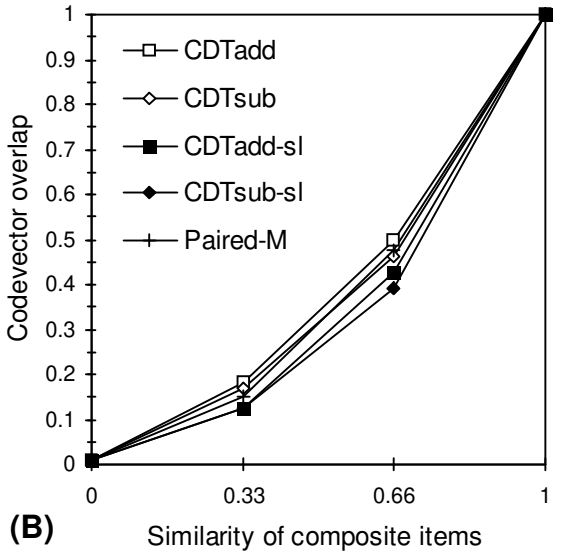


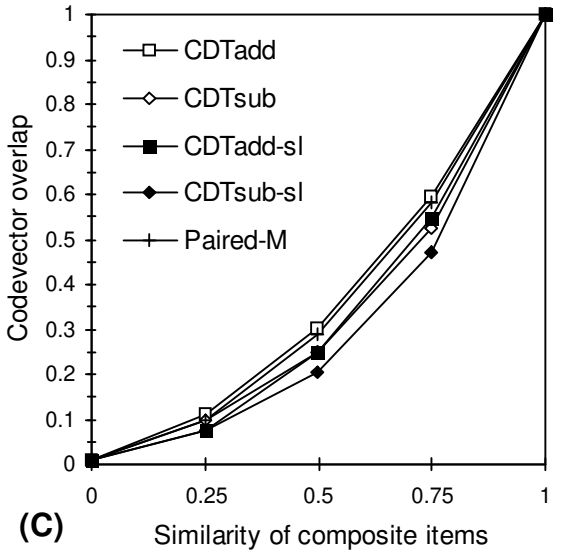
Figure 8. A neural network implementation of the subtractive Context-Dependent Thinning procedure. F_{out} executes conjunction of the input pattern in F_{in1} with inverted disjunction of K permutations of the input pattern. (Only two bundles of permutative connections are shown here: one by solid lines, and one by dotted lines). As follows from Table 3, K is approximately the same for the number S = 2,...,5 of component codevectors of certain density p.


CDTadd
CDTsub
CDTadd-sl
CDTsub-sl
Paired-M
Codevector overlap
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0.5
1
Similarity of composite items

(A)


CDTadd
CDTsub
CDTadd-sl
CDTsub-sl
Paired-M
Codevector overlap
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
0.33
0.66
1
Similarity of composite items

(B)


CDTadd
CDTsub
CDTadd-sl
CDTsub-sl
Paired-M
Codevector overlap
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
0.25
0.5
0.75
1
Similarity of composite items

(C)

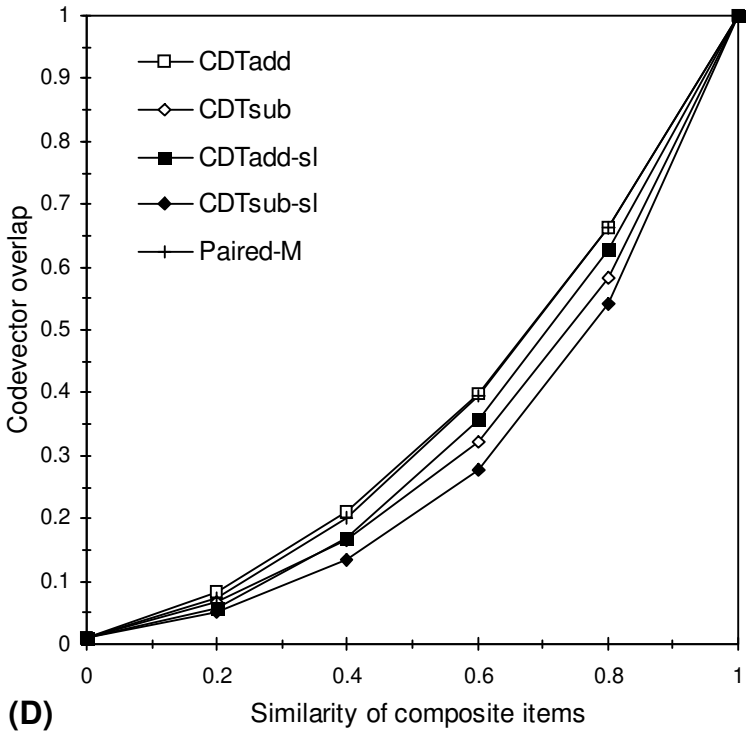
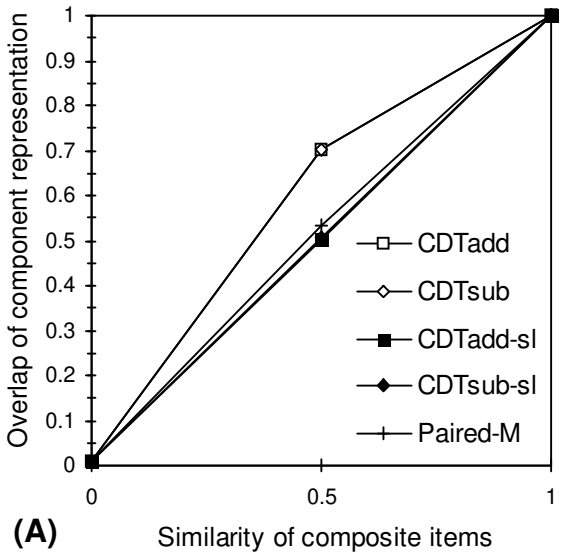
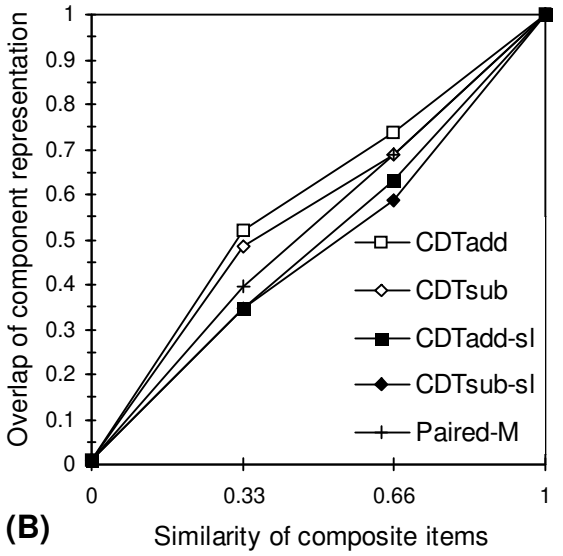


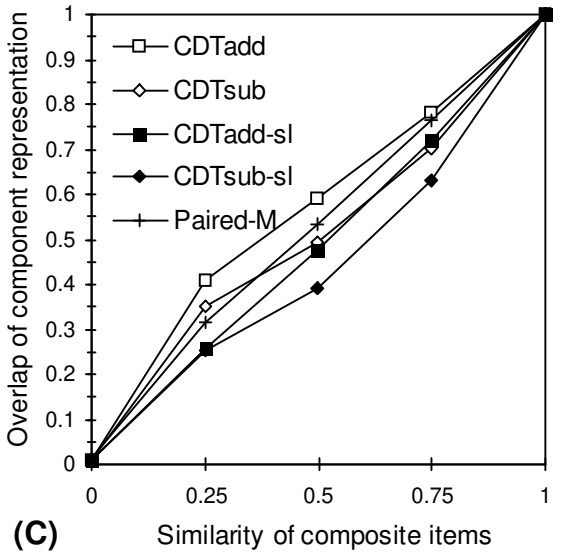
Figure 9. Overlap of thinned codevectors of composite items versus the similarity of the corresponding composite items for various versions of thinning procedures. CDTadd - the additive, CDTsub - the subtractive, CDTadd-sl - the selfless additive, CDTsub-sl - the selfless subtractive CDT procedures. Paired-M - paired conjunctive thinning; the densities of constituent codevectors are chosen to obtain M of 1s in the thinned codevector. For all codevectors, N=100000, M≈1000. The number of constituent codevectors: (A) 2; (B) 3; (C) 4; (D) 5.


Overlap of component representation
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.5
1
CDTadd
CDTsub
CDTadd-sl
CDTsub-sl
Paired-M
Similarity of composite items

(A)


Overlap of component representation
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.33
0.66
1
CDTadd
CDTsub
CDTadd-sl
CDTsub-sl
Paired-M
Similarity of composite items

(B)


Overlap of component representation
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.25
0.5
0.75
1
CDTadd
CDTsub
CDTadd-sl
CDTsub-sl
Paired-M
Similarity of composite items

(C)

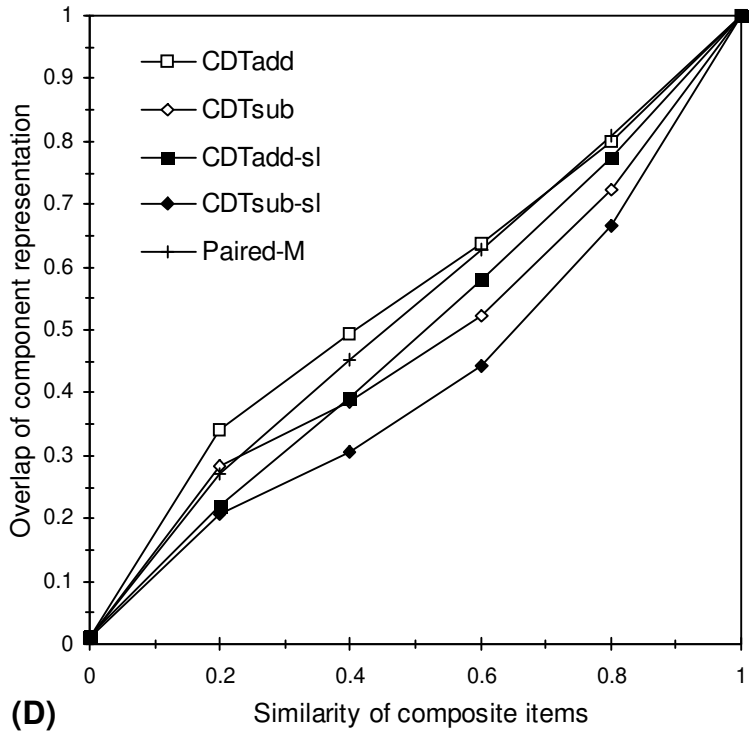
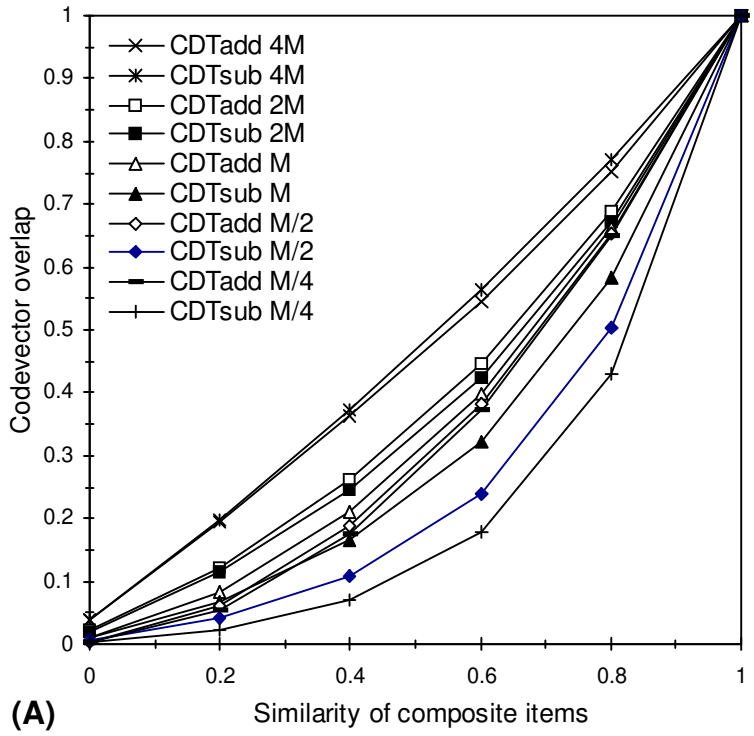


Figure 10. Overlap of constituent representations in thinned codevectors of composite items versus the similarity of the corresponding composite items for various versions of thinning procedures. CDTadd - the additive, CDTsub - the subtractive, CDTadd-sl - the selfless additive, CDTsub-sl - the selfless subtractive CDT procedures. Paired-M - paired conjunctive thinning; the densities of constituent codevectors are chosen to obtain M of 1s in the thinned codevector. For all codevectors, N=100000, M≈1000. The number of constituent codevectors: (A) 2; (B) 3; (C) 4; (D) 5.



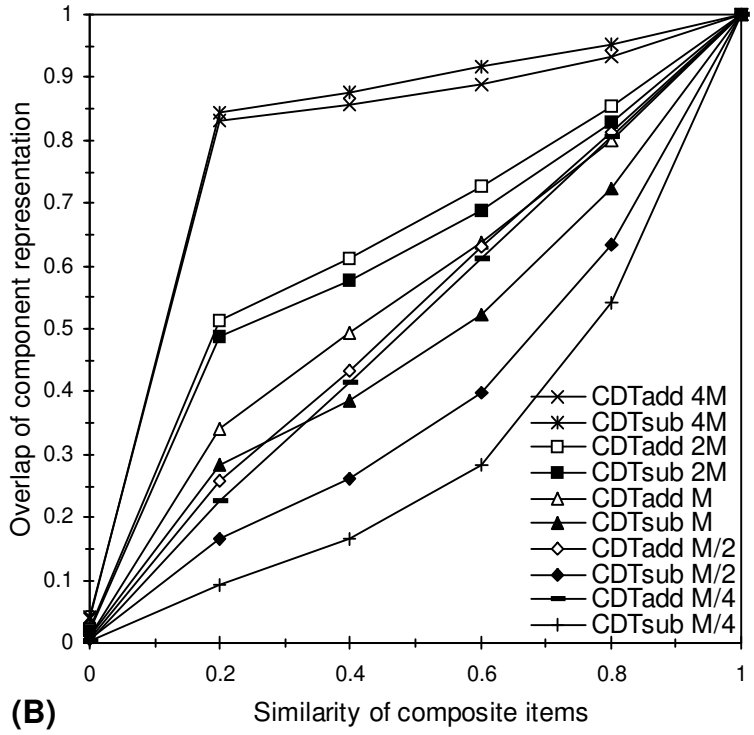


Figure 11. (A) Overlap of thinned codevectors and (B) Overlap of constituent representations in thinned codevectors for various "depth" of additive (CDTadd) and subtractive (CDTsub) procedures of Context-Dependent Thinning. There are five constituents in the composite item. Therefore the input codevector includes 5M of 1s. The resulting codevector was thinned to have from 4M to M/4 of 1s. For all codevectors, N=100000, M≈1000.