2014

Modular Composite Representation

J. Snaider

S. Franklin

Recommended Citation

Snaider, J., & Franklin, S. (2014). Modular Composite Representation. 10.1007/s12559-013-9243-y

Modular Composite Representation

Javier SNAIDER[1], Stan FRANKLIN[2]

Computer Science Department & Institute for Intelligent Systems, The University of Memphis

[1]FedEx Institute of Technology #403h, 365 Innovation Dr., Memphis, TN 38152;
E-mail: jsnaider@memphis.edu
[2]FedEx Institute of Technology #312, 365 Innovation Dr., Memphis, TN 38152

Abstract. High-dimensional vector spaces have noteworthy properties that make them attractive for representation models. A reduced description model is a mechanism for encoding complex structures as single high-dimensional vectors. Moreover, these vectors can be used to directly process complex operations such as analogies, inferences, and structural comparisons. Also, it is possible to reconstruct the whole structure from the reduced description vector. Here we introduce the Modular Composite Representation (MCR) a new reduced description model that employs long integer vectors. We also describe several experiments with them and give a theoretical analysis of the distance distribution in this vector space and of properties of this representation. Finally, we compare MCR with other two reduced description models: Spatter Code and Holographic Reduced Representation.

Keywords. Reduced Description, Holistic record, high-dimensional representation

1. Introduction

Distributed representations [1] are common in connectionist models, where the pattern of activation of their units (for example neurons in an artificial neural network) constitute its current representation, or in other vector representations, such as Latent Sematic Analysis (LSA) [2], where a high dimensional vector may represent a word. Frequently, in these representations each of their dimensions or units does not have any single specific meaning or interpretation, but the whole vector does. A nice property of these models is that in general they tolerate some amount of noise in their representations, allowing approximate comparisons. When a distributed representation is in the context of a high-dimensional vector space, other interesting properties arise. The distribution of the distances between vectors in these spaces, and the huge number of possible vectors, allow a noise-robust representation model where the distance between vectors indicates the similarity (or dissimilarity) of the concepts they represent.

However, a recurrent criticism of these representations is that it is difficult to represent *complex structures*, such as hierarchies, sequences, or graphs, or to perform high-level cognitive operations, such as analogies or logic inferences. Such complex structures and high-level operations are common in classic AI problems, (and are addressed using AI languages such as PROLOG), but also in other not so classic AI applications, such as cognitive architectures, robot controllers, and natural language processing. For example, in many cognitive architectures (e.g. [3, 4]) it is necessary to represent objects, events, actions, rules, and other pieces of data. The same is true for several natural language processing systems that require the representing not only of words, but of sentences, paragraphs, and whole documents as well. It is not easy to capture the rich structure of these kinds of data with plain vectors.

A reduced description model [5] is a mechanism for encoding complex structures as single vectors in the context of a distributed representation, where the dimensions of the vectors do not have any single specific meaning. The main idea behind reduced descriptions is to have a dual representation: the complex structure can be represented explicitly, with a vector representing each component in the structure, called a *full representation*, or as a *reduced description*, where a single vector represents the whole structure. See Figures 1 and 2 below for graphical conceptualizations of this process, and Section 6 for an analytical example. Also, reduced description vectors can be used to directly process complex operations such as analogies, inferences, and structural comparisons. Moreover, it is possible to reconstruct the full representation from the reduced description vector. Note that these representations preserve the nice properties of noise robustness and approximate comparisons of high dimensional distributed representations, and in addition, can perform some of these more complex operations. Both Plate [6, 7] and Kanerva [8] have presented several examples of the operations and applications that these representations allow. Kanerva presents simple cases of analogy making. Plate describes several experiments using reduced descriptions: storing sequences, similarity of shape configurations, and models for psychological analogy processing, among others. Stewart and colleagues produced an application based on spiking neurons that is capable of reasoning and planning, where the rules and system states are encoded using reduced description vectors [9]. For a comprehensive discussion of the possible uses of reduced description models see [7, 8].

Several authors [8, 10-12] have pointed out the importance of representations to perform tasks efficiently and solve problems. Winston defined the representation principle in these words: "Once a problem is described using an appropriate representation, the problem is almost solved" [12]. Franklin [10] discussed the importance of representation for both symbolic AI and connectionist models. Kanerva [8] pointed out how a representation can facilitate certain tasks at the expense of others. Even using the most advanced algorithms available today, several AI applications, such as cognitive architectures [3, 4, 13], robot controllers [14, 15], natural language processing [16-18], and Sentic Computing [19], perform poorly compared with humans when they try to solve problems that are generally easy for humans. These *challenging* AI applications can benefit from reduced description models that combine the robustness of high-dimensional vectors with the expressiveness of hierarchical and complex structures. For example, in addition to the already mentioned works of Kanerva, Plate, and Steward, Jones and colleagues [20] based BEAGLE, their semantic space model for natural language processing, on Holographic Reduced Representations, a reduced description model [7].

Here we introduce Modular Composite Representation (MCR): a new reduced description model that utilizes long integer vectors employing modular arithmetic in each dimension. This representation paradigm has properties similar to Spatter Code [21], which uses binary vectors, and to Holographic Reduced Representations (HRR) [6, 7], based on vectors of real or complex numbers. However, as we will show in the following sections, MCR has better performance than the other two models, simpler operations than HRR, and is more expressive than Spatter Code, making it a better option for most applications.

MCR satisfies the four desirable characteristics of reduced descriptions as prescribed by Plate [7]: *representation adequacy* (full descriptions can be reconstructed from the reduced descriptions), *reduction* (the reduced descriptions have a size similar to that of their components), *systematicity* (the process of constructing the reduced description from the full representation must be well known and deterministic), and *informativeness* (the reduced description encodes information about the full representation that it represents). MCR also provides *explicit similarity*; that is, similar elements[1] have similar representations.

Modular composite representation generalizes the ideas implemented in Spatter Code: the operations employed in MCR are equivalent to the XOR and integer sum defined in Spatter Code, but extended to the modular integer space. As Kanerva noted (personal communication with the authors), MCR also correlates with HRR in the frequency domain, which we will explore later in this paper.

In order to qualify as a reduced description representation model, MCR must define grouping and binding operations, as well as a similarity measure (or distance). The binding operation is the modular sum in each dimension, and the grouping operation is expressed as the vector addition of the equivalent vectors defined for each possible value in each dimension (see Section 5 for details). The distance employed for this model is a variation of the Manhattan distance, which takes into account the modular arithmetic of the vectors' values. This distance is explained and analyzed in detail in Section 4.

Figure 1 Here

Figure 1 shows an example using images to illustrate grouping and binding, the basic operations required for implementing a reduced description model. For representing the statement *"star first, circle second"*, we assign to the image A the meaning *first* and to the image B the meaning *second*. If we simply group the 4 images A, B, C, and D (in this case, performing an OR operation between them) we obtain figure E, which does not represent the required statement. On the other hand, if we first bind A and C, and B with D (in this example, using an AND operation between them), we obtain images F and G. Finally, grouping F and G produces image H that better represents the statement above.

This example is only intended to provide a conceptual idea of the binding and grouping operations; the images and operations that we use with them do not fully qualify as a reduced description model. Nevertheless, the example should help the reader to have a better conceptual understanding of the operations, and why they are necessary for the correct representation of structures.

The following sections review the basic concepts behind vector representations and reduced descriptions, and describe the vector space used in MCR, its basic operations, and its similarity measure.

[1] In the context of this work, we define elements as things that can be represented, for example, objects, actions, features, events, etc.

Next, we describe several experiments and compare their results with those of Plate's HRR. Then we analyze the expected value and variance of some expressions, and contrast MCR with Spatter Code and HRR. Finally, we discuss future research opportunities.

2. Vector Representations and Reduced Descriptions

In this section we discuss the main ideas behind vector representations and reduced descriptions. For further information, see [7, 8].

In many subfields of computer science, vector representations constitute one of the main types of data structure. For example, in machine learning, a vector of features–often of different data types–may denote a sample in a training set. A different approach, and closer to the focus of this work, utilizes vectors where all the dimensions share the same data type. Even with this uniformity, the number of possible representation models is limitless. The method of calculating or defining the vector that represents an element, and the distance or similarity measurement between these vectors, define the representation model and its properties. For example, in the last two decades a large number of semantic space models have emerged that use high dimensional vectors to represent words and texts. The most representative models include Latent Sematic Analysis (LSA) [2] based on statistical analysis; Random Indexing [16], which employs random sparse vectors and random permutations; and BEAGLE [20], which computes vectors using circular convolution. For recent surveys of semantic space models, see [17, 18].

In an even more generic view, vectors can represent any concept or element of interest: objects, features, rules, constraints, actions, etc. As explained above, when a vector belongs to a high dimensional space, interesting properties arise. For example, two randomly chosen points of the space are far away from each other, making them good candidates to represent unrelated concepts. As we will see, in these high dimensional vector spaces, the distances from a given vector to the rest of the vectors in the space tend to concentrate tightly around half of the maximum distance, which Kanerva calls the *indifference distance* [11].

As mentioned in the previous section, vector representations can implement what Plate [7] calls explicit similarity, where similar elements such as objects or concepts are represented with similar vectors. In other words, the vectors that represent similar concepts are close (i.e. the distance between them is small). This feature may facilitate some operations such as the retrieval of similar instances from a memory, the clustering of similar objects, and the matching against prototypes [7]. In contrast, other common techniques for referencing elements in computer science, such as pointers, or hashing, lack this advantage. Vectors can employ several kinds of similarity measures, e.g. the cosine, or the inverse of some distance, such as the Hamming (for binary vectors), the Euclidean, and the Manhattan distances.

Explicit similarity becomes even more advantageous when using vectors that belong to high dimensional spaces (i.e., vector spaces with a large number of dimensions). Such spaces offer an enormous number of possible combinations of values that the dimensions of the vectors can take (also known as patterns), making the necessity for compact representations less critical. There is no need for a one to one correspondence between patterns and elements. For example, in a binary space with 1,000 dimensions, we can theoretically represent 2^{1000} different elements, though this would be highly unlikely. We can use just a fraction of the vectors in the space, say 2^{100} vectors distributed in the space, which still allows a gigantic number of possible representations. Even after adding some noise by introducing a few changes in one of these vectors, it can still represent the same element. In other words, *a region of the space, instead of just one point, represents an element, creating a more noise robust representation that gracefully degrades as noise increases, and produces desirable properties such as pattern completion*.

Figure 2 here

As mentioned before, it is difficult to represent complex structures, such as sequences and hierarchies using distributed representations (or vector representations). Performing high level cognitive tasks such as reasoning, planning, or action selection often involves structures with multiple components. Implementations of these tasks frequently utilize structures such as sequences or hierarchies, and require variable binding. Moreover, the components of these structures can, in turn, be complex structures themselves. Of course, we can create these structures, and use vectors as components. But, in that case, the vectors become mere symbols, with a significant loss of expressive power. To address this problem, Hinton [5] introduced the concept of reduced description, a method for encoding complex structures as single vectors. The main idea is to have a dual representation: the structure can be represented explicitly (fully), with a vector for each component of the structure, or as a reduced description, where a single vector represents the whole structure. When the system focuses on a particular component, its constituent structure is represented in full, instantiating all the components (vectors) that compose it. On the other hand, when the component participates in the structure of another component that has the current focus, it is represented with a single vector as a reduced description. See Figure 2.

The reduced description is not a mere pointer to the full description, but a loosely compressed version of the original structure. Using pointers to create data structures has a long history in computer science. For example, a *struct* in the C programming language can have several fields, where some of them may also be pointers to other structures. Pointers help create lists, trees, or other data structures. Object oriented languages, such as Java, hide the pointers from the programmer by using object references, but they employ essentially the same mechanism: an object reference leads to the actual location of the object in memory. A pointer (or object reference) does not have any direct relationship with the data to which it points. In other words, looking at the pointer rather than its referent reveals nothing about the data. Furthermore, given a piece of data, or part of it, it is not possible to locate it easily. Hash indexing is probably the traditional computer science technique most similar to reduced descriptions. Hashing allows the location of data to be calculated from its content. However, the hashing usually does not provide any information about the content (not informative), and similar elements often have very dissimilar hashing values (lacks explicit similarity). The reduced descriptions, on the other hand, are abbreviated representations of the full data. Moreover, as we shall see, several operations can directly use the reduced descriptions without needing to recover the original data.

Plate [7] analyses reduced descriptions from four desirable characteristics, as mentioned in Section 1:

- *Representation adequacy*: The reduced description must enable the reconstruction of the full representation. Failing this is analogous to a pointer that does not point to its data. Such reconstruction is possible thanks to the noise robustness of the model, and with the help of a cleanup memory (see details below).
- *Reduction*: The reduced description must be smaller than the full representation. In general, the vectors used in vector representations are of a fixed size, and a single vector of the same size comprises a reduced description.
- *Systematicity*: The construction of the reduced description should be systematic. That is, the way to construct the reduced description from the full representation must be well known and deterministic. This facilitates the reconstruction of the full representation, when is required.
- *Informativeness*: The reduced description should contain some information about the whole it represents. This allows its direct use for certain operations without retrieving the full representation.

Moreover, reduced descriptions produce vectors that exhibit explicit similarity.

Many of the complex structures apply to AI problems pervasively; examples include sequences, hierarchies, and predicates (i.e. rules with variable binding). These structures, and probably others, can be constructed out of even simpler primitives such as binding and grouping. *Binding* is the assignment of one element, which is called the filler, to a particular role or position in the structure. For example, in a sentence, the element “Sue” can be bound to the *subject* role. *Grouping* forms a set of elements. For example, the structure representing a sentence can be a collection of roles bound to their fillers, where each role stands for a part of the sentence. In a similar way, a sequence can be modeled with the group of its elements, each of them bound to its position in the sequence. To create a reduced description model, we need to define binding and grouping operations[2], and a distance or similarity measure. Kanerva [8] introduced more abstract names for these operations; he uses multiplication for binding, and sum for grouping, which simplifies the operational notation. We will use this same convention here. The following paragraphs summarize Kanerva's ideas of hyperdimensional arithmetic [8].

In general, the multiplication and sum operations do not necessarily correspond with the usual arithmetic operations, but they should have several properties in common. These properties, in turn, facilitate the achievement of the four characteristics of reduced description models described above. For example, the multiplication must be reversible; this allows *unbinding* the filler to reconstruct the original structure. We will discuss several examples that illustrate the use of these properties in Section 6.

If a vector A represents an element and vector B represents a role, the binding of A to B is given by:

$$C = A \otimes B \tag{1}$$

where ⊗ denotes the multiplication operator (e.g., XOR in Spatter Code). Multiplication by the *inverse* vector reverses this operation. The definition of the inverse vector depends on the multiplication operator used:

$$A = C \otimes B^{-1} \tag{2}$$

In the binary case using XOR, $B^{-1} = B$.

The multiplication must be commutative and associative:

$$A \otimes B = B \otimes A \tag{3}$$

$$(A \otimes B) \otimes C = A \otimes (B \otimes C) \tag{4}$$

The multiplication also must preserve distances:

$$d(A, B) = d(A \otimes C, B \otimes C) \tag{5}$$

This property helps in the process of recovering the components of a reduced description vector. See below for details.

Interestingly, multiplication in general produces a vector that is dissimilar to its operands. That is, they are almost at the indifference distance. See Section 4 for details.

[2] Some systems can create reduced descriptions without explicitly defining these operations. For example see RAAM [22].

$$A \otimes B \not\approx A \; and \; A \otimes B \not\approx B \tag{6}$$

where ≉ denotes dissimilarity.

The sum must be commutative (7), and it is desirable that it also be associative (8):

$$A + B = B + A \tag{7}$$

$$(A + B) + C = A + (B + C) \tag{8}$$

However, this last condition is difficult to achieve, and most models do not satisfy it [7, 8].

The most important property of the sum is that it produces a resultant vector similar to its operands.

$$A + B \approx A \; and \; A + B \approx B \tag{9}$$

This property is especially useful for finding the operands in an associative memory given the resultant. In the same way, it is possible to retrieve the sum vector from the memory with a partial set of the operands. See below for more details.

In general, the sum operation produces an intermediate result, that is a vector outside the domain of the space, and a normalization process is required to yield the final result. This normalization is in general the cause of a lack of associativity. To mitigate this problem, a multi-operand sum may be defined that first computes the intermediate result combining all the operands, and normalizes it (denoted by [...]) at the end.

$$Sum(A, B, C, \dots) = [A + B + C + \cdots] \tag{10}$$

Finally, multiplication has to distribute over sum:

$$A \otimes (B + C) = A \otimes B + A \otimes C \tag{11}$$

This property will serve well for extracting the components of a reduced description vector.

Random permutations (denoted by capital Greek letters: Π, Γ, etc.) can be used as a kind of multiplication. This is not a real multiplication, because one of the operands is not a vector, but different permutations can represent different roles. In this case, applying a permutation to a vector binds the vector to the role represented by the permutation. For example, if Π and Γ represent color and shape respectively,

$$A = \Pi(red) + \Gamma(square) \tag{12}$$

then A represents a red square.

Permutations also preserve distances, are commutative, associative, and distributive over the sum. Moreover, they have an interesting advantage over other multiplications: they preserve the vectors' *density*, defined as the relative number of zeros and ones. Some associative memories (e.g., [23]) and some representation models (e.g., [24]) perform better with sparse vectors (i.e., vectors with few ones). Permutations work well for both sparse vectors and dense vectors, which have a relatively equal number of zeros and ones. See [24] for further discussion of this subject. Finally, combining random permutations with some multiplications yields a non-commutative multiplication that is useful for modeling some structures, for example, sequences. We will briefly discuss such use in Section 5. For more details see [25].

Summing up, to create a reduced description we have to define multiplication and sum operations, as well as a distance measure in a vector space. The multiplication must be associative, commutative, and distributive over the sum. It must also preserve distance, and produce vectors dissimilar to its operands. The sum has to be commutative, and must produce vectors similar to its operands. These properties of the multiplication and the sum allow the creating of reduced description vectors, and the performing of operations described later in the hyperdimensional section (Section 6). A discussion of these properties can be found in [7, 8].

3. Modular Integer Vectors

MCR utilizes large modular integer vectors. These vectors have a defined integer range of possible values for each dimension. For example, the range of values can be {–8, 7} or {0, 15}. Although any range of values is possible, for simplicity in the notation and analysis, we will use ranges with 0 as the lower bound and r – 1 as the upper bound, only even values of *r*, and will give all dimensions the same range of values. In more formal notation, MCR employs vectors within a multidimensional space, $v \in \mathbb{Z}_r^n$, where *n* is the number of dimensions of the space and *r* is the size of the range of values for each dimension. The values of vectors in each dimension employ modular arithmetic. The greatest possible value for a dimension is *r* – 1 and the next value after *r* – 1 is 0.

Modular integer vectors define a homogeneous space, i.e. any vector of the space can be transformed into any other by an invertible linear transformation, a linear isomorphism. In other words, the space "looks" the same to any of its vectors. Using normal (i.e. non-modular) integer vectors with values in a range for each dimension does not have the same property. A simple example may clarify this: in a two dimension space, with $r = 10$, the vector (0, 0) does not have the same view of the space as the vector (5, 5). The former vector has fewer neighbors than the latter. On the other hand, if the space uses modular arithmetic, it has no borders, and each point has an equivalent point of view of the space.

Figure 3 serves to clarify the following definitions of possible relations between values. The *complement* of a value is another value such that their sum equals *r*. For example the complement of 3 is 13. The *opposite* of a value is the value in its *antipode*, which is calculated by adding *r*/2 to it.

Figure 3 here

Several integer arithmetic operations have their corresponding modular versions. The modular sum corresponds to the arithmetic sum modulo *r*:

$$a_r + b_r = mod_r(a + b) \tag{13}$$

where $mod_r(\ldots)$ is the reminder of the integer division by *r*. For example, if $r = 16$, the modular sum of 6 and 12 is 2. The modular subtraction is defined in a similar way:

$$a_r - b_r = mod_r(a - b) \tag{14}$$

Subtraction can also be expressed as the sum of the complement. To show this we can add *r* inside the mod_r term, which does not alter the result:

$$a_r - b_r = mod_r(r + a - b) \tag{15}$$

or

$$a_r - b_r = mod_r(a + (r - b)) \tag{16}$$

where $(r - b)$ is the complement of *b*. Other operations such as multiplication and division also have equivalents in modular arithmetic, but MCR does not utilize them.

The individual values in each dimension of the vectors used in MCR do not have to follow any particular statistical distribution, however, for the analyses in this paper to be valid, we assume that they follow a uniform distribution in the range $\{0, r - 1\}$. In contrast, HRR vectors must follow a normal distribution with specific parameters; otherwise, the operations defined in HRR to combine vectors do not produce the desired results [7].

It is desirable that the distance between vectors correlates with the distance between the concepts that they represent. To this end, unrelated concepts can be represented by random vectors that are uniformly distributed across their high dimensional space, since such vectors tend to be far apart (at the indifference distance) from each other [11].

4. Manhattan Distance in a Modular Space

MCR utilizes a variation of the Manhattan distance defined as

$$d(u,v) = \sum_{i} \Delta_i \tag{17}$$

where

$$\Delta_i = \min\big(mod_r(u_i - v_i), mod_r(v_i - u_i)\big). \tag{18}$$

This variation of the Manhattan distance takes into account the modular characteristic of the space. For example, if we consider a vector of only one dimension, with possible values between 0 and 15, the above-defined distance between the vector 0 and the vector 14 is 2 (because $mod_{16}(-14) = 2$), whereas using the normal Manhattan distance, the distance between these vectors would be 14. See Figure 3 for reference. As we will see, in high dimensional vector spaces, the distances from a given vector to the rest of the vectors in the space tend to concentrate tightly around half of the maximum distance, which we have called the indifference distance.

In order to analyze the behavior and properties of the modular integer vectors employed in MCR, it is useful to know the distribution of the distances among the vectors in the space. The following theorem approximates this distribution for the case when *r* is even. The result is similar, but not exactly the same, when *r* is odd. For this analysis, the dimensions of each vector are considered to contain independent random variables that are uniformly distributed in $\{0, r-1\}$. This same theorem was introduced in [26] and is reproduced here for completeness.

Theorem:

If the dimensions of all vectors are independent and uniformly distributed in $\{0, r-1\}$ and *r* is even, then the distribution of Manhattan distances from a given vector to the rest of the vectors of the space can be approximated by the following normal distribution:

$$D \sim N\left(\frac{nr}{4}, \frac{n(r^2+8)}{48}\right) \tag{19}$$

Proof. The dimensions of the vectors are independent and uniformly distributed in $\{0, r-1\}$. The distance from the origin to a vector $v \in \mathbb{Z}_r^n$ will be the sum of n random variables $X_i = \Delta_i$, where $\Delta_i = \min\big(mod_r(0 - v_i), mod_r(v_i - 0)\big)$.

The possible values of X_i are between 0 and $r/2$, and X_i does not have a uniform distribution since values 0 and $r/2$ have half of the probability of the other possible values. This is due to the modular property of the space and the distance calculation. For example, if $r = 16$, the maximum difference in dimension i between v and the origin is 8, and the only possible value of v_i is 8. The same is true for a difference of 0. For other possible values of the difference, for example 4, there are 2 possible values of v_i: 4 and 12. More formally, since adding r to the argument of the mod_r function does not alter the result, we can rewrite the expression of X_i as

$$X_i = \min\big(mod_r(r - v_i), mod_r(v_i)\big) \tag{20}$$

The values of v_i are uniformly distributed in $\{0, r-1\}$. If $v_i = 0$, then both arguments of the min function are zero; thus $X_i = 0$. For all other possible values of v_i none of the arguments of min is zero, thus $v_i = 0$ is the only value that produces $X_i = 0$, and then $P(X_i = 0) = 1/r$.

For values of $v_i \in \{1, r-1\}$ the argument of the two mod_r functions are positive and less than r. So, we can rewrite the expression of X_i as

$$X_i = \min(r - v_i, v_i) \text{ where } v_i \in \{1, r-1\} \tag{21}$$

It is easy to see that the maximum value of $X_i = r/2$. If $v_i \leq r/2$, then $r - v_i \geq r/2$, and then $X_i = v_i$, which is less than or equal to $r/2$. On the other hand, if $v_i \geq r/2$, then $r - v_i \leq r/2$, and then $X_i = r - v_i$ which is less than or equal to $r/2$. Notice also that for $X_i = r/2$, either $r - v_i = r/2$ or $v_i = r/2$. But, $r - v_i = r/2$ implies that $v_i = r/2$. Thus, only this value produces $X_i = r/2$, and then $P(X_i = r/2) = 1/r$.

Finally, each value $x \in \{1, r/2 - 1\}$ of X_i is produced by exactly two values of v_i. In effect,

$$x = \min(r - v_i, v_i) \Rightarrow r - v_i = x \; or \; v_i = x \text{ where } 1 \leq x \leq r/2 - 1 \tag{22}$$

Following reasoning similar to that of the previous paragraph, it is clear that exactly one value of v_i less than $r/2$ and one greater that $r/2$ satisfy the second half of the previous expression for each value of x such that $1 \leq x \leq r/2 - 1$. Then, $P(X_i = x) = 2/r$, where $1 \leq x \leq r/2 - 1$.

Summing up, the distribution of X$_i$ follows

$$P(X_i = x) = \begin{cases} \frac{1}{r} & x = 0, \frac{r}{2} \\ \frac{2}{r} & 1 \leq x < \frac{r}{2} - 1 \\ 0 & otherwise \end{cases} \tag{23}$$

Since the distribution of X$_i$ is symmetric on {0, *r*/2}, the expected value of X$_i$ is half of its maximum possible value, or *r*/4. The variance of the distribution of X$_i$ requires some more analysis. We introduce the simplifying substitution, *r*’ = *r*/2. Then, the variance of X$_i$ will be

$$\sigma^2 = \frac{1}{r'}\sum_{i=1}^{r\prime-1}\left(i - \frac{r'}{2}\right)^2 + \left(\frac{1}{2r'}\left(0 - \frac{r'}{2}\right)^2\right) + \left(\frac{1}{2r'}\left(r' - \frac{r'}{2}\right)^2\right) \tag{24}$$

$$\sigma^2 = \frac{1}{r'}\sum_{i=1}^{r\prime-1}\left(i - \frac{r'}{2}\right)^2 + 2\left(\frac{1}{2r'}\left(0 - \frac{r'}{2}\right)^2\right) = \frac{1}{r'}\sum_{i=0}^{r\prime-1}\left(i - \frac{r'}{2}\right)^2 \tag{25}$$

$$\sigma^2 = \frac{1}{r'}\sum_{i=0}^{r\prime-1}\left(i^2 - ir' + \frac{r'^2}{4}\right) = \frac{1}{r'}\sum_{i=0}^{r\prime-1}(i^2) - \sum_{i=0}^{r\prime-1}(i) + \frac{r'^2}{4} \tag{26}$$

$$\sigma^2 = \frac{(r'-1)(2r'-1)}{6} - \frac{(r'-1)r'}{2} + \frac{r'^2}{4} = \frac{r'^2+2}{12} \tag{27}$$

Substituting back *r*, the variance of X$_i$ is

$$\sigma^2 = \frac{r^2+8}{48} \tag{28}$$

Since X$_1$,…, X$_n$ are independent and identically distributed and

$$D = \sum_{i=1}^{n} X_i \tag{29}$$

it follows from the central limit theorem that for large number of dimensions n, we can approximate the distribution of the distances by a normal distribution with mean nE[X$_i$] and variance var(X$_i$)n. In conclusion, the distribution of distances from the origin (or any other point) to the rest of the points of the space is:

$$D \sim N\left(\frac{nr}{4}, \frac{n(r^2+8)}{48}\right) \tag{30}$$

which proves the theorem □.

When n is large, for example 1,000 or 10,000, the ratio between the mean and the standard deviation of the distance distribution tends to be large, with values concentrated around half of the maximum distance. For example, when n = 1,000 and r = 16, the distribution of the distances is well-approximated by a normal distribution with a standard deviation of 74.16 and mean distance of 4,000. Dividing the mean by the standard deviation–about 54 in this example–yields the number of standard deviations between a vector and the bulk of the space. Notice that per the normal distribution, 99.9999% of the vectors of the space lie within five standard deviations of the mean, corresponding to distances between 3,630 and 4,370 in the current example. The probability of a random vector of being closer than 3,000 is almost zero ($\sim 10^{-43}$), which is a useful property that helps to make the model extremely robust.

5. Basic Operations of MCR

The two basic operations, grouping and binding, constitute the heart of the reduced description models. Grouping (or sum) operation is used to create sets or groups of elements, and binding (or multiplication) creates representations for bonds among elements, such as in the role-filler case. Given that the required properties of these operations are responsible for the behavior and characteristics of the reduced description models, each model can define these operations according to the characteristics of its vector space. In this way, we can abstract the reduced description model ideas and its basic operations to explore problems and applications independently of the reduced description implementation. For example, consider the following expression that represents a red circle:

$$F = [circle \otimes Shape + red \otimes Color] \tag{31}$$

where *circle*, *Shape*, *red*, and *Color* are vectors, and the symbols ⊗ and + represent the binding and grouping operations respectively. This expression can work in any reduced description model with appropriate definitions for grouping and binding.

The rest of this section defines the binding and grouping operations used in MCR. These definitions fulfill all the requirements described above, enabling MCR to serve as a reduced description system capable of performing hyperdimensional computations. Kanerva [8] and Plate [6, 7] introduce many applications of hyperdimensional computing.

The binding (or multiplication) of modular integer vectors is defined as the modular sum in each dimension. For example, the multiplication of two vectors A and $B \in \mathbb{Z}_{16}^{n}$, with values for dimension *i* equal to 10 and 12 respectively, produces a new vector *C* with the value of dimension *i* equal to 6.

$$C_i = mod_r(A_i + B_i) \tag{32}$$

This operation resembles the bitwise XOR used in Spatter Code.[3]

The unbinding operation is simply the modular subtraction in each dimension, or the modular sum of the first operand with the complement of the second operand in each dimension. This leads to the definition of the inverse vector in this model. The inverse of the vector *A* is another vector A^{-1} such that the value of each dimension *i* of A^{-1} is the complement of the value of *A* in the same dimension:

$$A_i^{-1} = mod_r(r - A_i) \tag{33}$$

This multiplication operation has all the properties described above in the reduced description section: It is associative, commutative, distributive over the sum (see below), and preserves distances. Given that the definition of MCR vector multiplication employs the modular sum in each dimension, it inherits its associativity and commutativity properties. For example, when adding the values of dimension *i* of two vectors, $mod_r(A_i + B_i) = mod_r(B_i + A_i)$. Also, for this operation it holds that

$$mod_r\big(A_i + mod_r(B_i + C_i)\big) = mod_r(mod_r(A_i + B_i) + C_i) \tag{34}$$

These properties also lead to the distance-preserving property of this multiplication.

Theorem

The multiplication of MCR vectors defined above preserves the distance between vectors. Given three MCR vectors *A*, *B*, and *C*, the following equality holds:

[3] Actually, XOR is a special case of the modular sum when *r* = 2.

$$d(A, B) = d(A \otimes C, B \otimes C) \tag{35}$$

Proof. Suppose the distance between *A* and *B* is *d*. From equations (17) and (18)

$$d = \sum_i \min\big(mod_r(A_i - B_i), mod_r(B_i - A_i)\big) \tag{36}$$

After multiplying *A* and *B* by *C*, the first operand of the *min* function becomes

$$mod_r\big(mod_r(A_i + C_i) - mod_r(B_i + C_i)\big) \tag{37}$$

Applying the associative and commutative properties of the modular sum produces the following expression:

$$mod_r\big(mod_r(A_i - B_i) + mod_r(C_i - C_i)\big) = mod_r(A_i - B_i) \tag{38}$$

which is identical to the original expression before the multiplication. Applying the same procedure to the second operand produces a similar result. Consequently,

$$d(A, B) = d(A \otimes C, B \otimes C) \tag{39}$$

which proves the theorem □.

This multiplication produces vectors that tend to differ from the operands.

$$A \otimes B \not\approx A \text{ and } A \otimes B \not\approx B \tag{40}$$

Later we will explore the expected value and variance of the vectors produced by the multiplication.

In some cases, a non-commutative multiplication comes in handy. For example, such a multiplication can be used to distinguish unambiguously between the filler and the role in a binding operation. Applying a random permutation by changing the order of the dimensions of one of the operands before computing the multiplication produces an alternative non-commutative multiplication. For more details, see [7, 8]. In general, a particular system will employ a single random permutation that does not change for that particular system after its creation.

The grouping (or sum) operation is a bit more difficult to define. In fact, there are several options for this operation. To correctly evaluate the different options, we have to consider that *producing vectors similar to its operands is the most important characteristic of the grouping operation*. This similarity allows identifying a composed vector from some of its components, and vice versa. This is a fundamental property of reduced description models. The first alternative for the sum consists of using the *average* of the operands' values for each dimension, choosing randomly among the closest ones if the average produces a non-integer value. This value corresponds to the middle point on the arc between the two values corresponding to each operand on the circle of Figure 3. For example, if we group the vectors A and $B \in \mathbb{Z}_{16}^{n}$ with values for dimension i being 10 and 12 respectively, the result has a value 11 in that dimension. Applying this operation to all dimensions produces a new vector that is approximately equidistant from its operands. A problem arises when the vectors to group have opposite values for one dimension, since the average then has two possible values that must be defined by chance. For example, the average for a particular dimension of vectors with values 5 and 13 can be either 9 or 1.

The lack of associativity in the averaging operation generates further difficulties when grouping several vectors, as illustrated in the following example. In the same modular space with r = 16, the average of values 0, 7 and 8 yields 5; however, averaging 7 and 8 first and then grouping with 0 produces a different result (4). Associating the values in other ways produces yet other results. Even worse, if the values of the operands lie in different semicircles (see Figure 3), the average must consider the two possible paths between values (i.e. the two arcs on the circle that connect the values in one direction or another), picking the one that minimizes the distances from the resulting value to the operands, overcomplicating the operation.

Another interesting solution to defining the grouping (sum) operation utilizes a mechanism similar to the sum operation defined for HRR in the frequency domain [7]. Let us consider each possible value of a dimension as a vector of unit length in a plane, called an *equivalent vector*. The center of the circle in Figure 4 corresponds to the coordinate's origin in this plane. For example, the equivalent vector for the value zero is (0, 1) and the value seven corresponds to the equivalent vector $(\sqrt{2}, -\sqrt{2})$. This sum operation involves two steps to calculate the value of each dimension i: the equivalent vector sum and the normalization. The first step consists of calculating the rectangular sum (i.e. their vector sum) of the equivalent vectors corresponding to the values of each operand for dimension i. Second, the normalization process calculates the value of each dimension of the group vector as the closest value corresponding to the resultant vector normalized to length one. Since the dimensions have only r possible values, a table with the equivalent vectors' components and the tangent of their angles can speed up the calculation and normalization processes. Figure 4 shows the representation of the equivalent vectors and a couple of examples of grouping.

We can attach a weight to some of the vectors when we group them by multiplying the corresponding vectors of their dimension values by a scalar or weight. For example, suppose we want to group the vectors A and B with weights w_A and w_B . For each dimension i we have to sum the equivalent vectors a_i

and b_i corresponding to the values A_i and B_i respectively, multiplying a_i by the scalar w_A and b_i by the scalar w_B, or

$$C_i = \text{value}_r(w_A a_i + w_B b_i) \tag{41}$$

where value$_r$ (x) produces the closest value corresponding to the vector x.

We can extend the definition of this sum for the case of more than two operands by simply summing, in each dimension, all the equivalent vectors of the operands for each dimension before normalizing. Grouping several operands in this way produces more consistent results than summing and normalizing in each individual group operation between two operands. Figure 4 depicts the result for combining three vectors that have values 0, 7, and 13 respectively for a given dimension.

Interestingly, the length of the resultant vector gives an idea of the quality of the resulting value for that dimension: a longer resultant vector is more likely to represent an almost mid-point between the operands' values than a shorter one. Similar values have equivalent vectors with similar directions. Adding these equivalent vectors will produce a new vector with length approximately equal to the sum of the operands' lengths. On the other hand, a short resulting vector indicates that several opposite (or near opposite) equivalent vectors comprise the operands, producing a resulting vector dissimilar to some (or all) of these values. Figure 4 illustrates examples of both situations. Finally, it is worthy of mention that using this definition of sum produces the same result as the average version in the case of grouping only two vectors.

The final option for grouping is similar to the one used in Spatter Code [8]: applying a majority rule in each dimension. This simple technique works only when combining several vectors because with few operands, the chances of equal values in one dimension in several vectors is small, producing an undefined value in that dimension that must be determined randomly.

Comparing these options for the grouping operation, clearly the sum of equivalent vectors emerges as the most appropriate one. The other options have serious flaws, including more complex algorithms, or the introduction of more noise in the result. When combining only two vectors, the average of each dimension, which produces the value corresponding to the midpoint of the shorter arc between the two values in the circle of values, is still useful due to its simplicity. The complexity of the sum, defined as the addition of equivalent vectors, is $O(nt)$ where n is the number of dimensions of the vector and t is the number of vectors to group. However, this operation requires calculating the components of the vectors representing the values of each operand and each dimension, which involves calculating the sine and the cosine of the angle of the equivalent vector of each value, and an arctangent at the end, which could be computationally expensive (i.e. a large constant in the time complexity). Nevertheless, since there are only *r* possible and predefined values for each dimension, using tables for the two components and the tangent of the equivalent vectors mitigates this problem.

This addition of equivalent vectors grouping operation has the required properties described in previous sections. Since the rectangular sum of vectors is commutative, the grouping operation shares this property. This operation is not associative because of the normalization after each sum. However, using the expanded definition for several operands as defined above ameliorates this problem. The sum vector is similar to its operands. This property is analyzed in detail in Section 8. Finally, the multiplication distributes over the sum. We can interpret the multiplication as a *rotation* of the circle of values for each dimension. Clearly, rotating equivalent vectors and then adding them produces a resulting vector identical to the result of first adding the equivalent vectors and then rotating.

6. Hyperdimensional Computing with Composite Modular Representation

In this section, we will use an example of encoding events with MCR, which Plate [7] introduced when presenting HRR, allowing us to compare the results from both models. This example employs 512-dimensional vectors with an *r* of 16.

Some hyperdimensional operations produce noisy versions of the target vector, requiring a cleanup memory with all the vectors used in the experiment to produce the correct vector. When required, this example will use a hash table data structure to maintain all the vectors, and an exhaustive search procedure that computes the distances from a given vector to all the vectors in the table, returning the closest ones. At the end of this section, we present the results from the same experiments using Integer Sparse Distributed Memory (ISDM) [26, 27] as cleanup memory.

We follow Plate's example, defining the same vectors as he did. First we create some base vectors (vectors representing features of which other vectors are composed) choosing randomly the values for each of their dimensions. Consequently, these vectors are independent and uniformly distributed in the space. The expected distance between these vectors is around the mean distance (or indifference distance) $nr/4$ (2,048 in this example). Composing some of these base vectors by grouping and binding them defines the vectors that represent more complex elements. For clarity, base vectors will be divided into three categories: event types, object features, and role features. The event type category includes the vectors **cause**, **eat**, and **see**. The object feature category comprises **being**[4], **human**, **state**, **food**, **fish**, and **bread**. Finally, **object** and **agent** constitute the role features group. The following formulas define the token and role vectors for this example:

[4] This vector could have been omitted, but we chose to follow Plate's example that included it.

$$\mathbf{mark} = \mathbf{being} + \mathbf{human} + \mathbf{id}_{mark} \tag{42}$$

$$\mathbf{john} = \mathbf{being} + \mathbf{human} + \mathbf{id}_{john} \tag{43}$$

$$\mathbf{paul} = \mathbf{being} + \mathbf{human} + \mathbf{id}_{paul} \tag{44}$$

$$\mathbf{luke} = \mathbf{being} + \mathbf{human} + \mathbf{id}_{luke} \tag{45}$$

$$\mathbf{thefish} = \mathbf{food} + \mathbf{fish} + \mathbf{id}_{the\_fish} \tag{46}$$

$$\mathbf{thebread} = \mathbf{food} + \mathbf{bread} + \mathbf{id}_{the\_bread} \tag{47}$$

$$\mathbf{hunger} = \mathbf{state} + \mathbf{id}_{hunger} \tag{48}$$

$$\mathbf{thirst} = \mathbf{state} + \mathbf{id}_{thirst} \tag{49}$$

$$\mathbf{eat}_{\boldsymbol{agt}} = \mathbf{agent} + \mathbf{id}_{eat\_agent} \tag{50}$$

$$\mathbf{eat}_{\boldsymbol{obj}} = \mathbf{object} + \mathbf{id}_{eat\_object} \tag{51}$$

Other role vectors, such as $\mathbf{see}_{agt}$, have similar definitions. Notice that these are just examples of operations among the vectors. Depending on the problem, other formulas for defining vectors can be used. In this case, the construction of these vectors using these expressions produces similar vectors within each category, which are also dissimilar to vectors in other groups. For example, in Table 1 the vectors **mark** and **paul** are similar, and both are dissimilar to **thebread**. The id vectors are also random vectors (generated in the same way as the base vectors) that help to discriminate the vectors within the same group. We can considerer **fish** as a **being**, and construct the **fish** vector accordingly, but we follow Plate's example where he defined the **fish** vector with the expression above.

Table 1 summarizes the distances among representative vectors in the example. The distance between a vector and itself is always zero. Notice that in HRR, this is not always the case [7]: a vector can have a distance from itself different than zero. (However, in the HRR frequency domain, this distance is always zero.)

Vectors with common features, such as vectors that represent persons, have small distances between them. According to equation (19), the distance distribution of the vectors in the space has a *SD* approximately equal to 50 and a mean of 2,048. The likelihood that **mark** and **john** are within distance 1,078 of each other by chance alone is almost zero (~10^{-69}). The distances among unrelated vectors cluster around 2,048, the indifference distance.

Table 1 here

Table 2 here

Using the token and role vectors, we can create vectors representing different events. Table 2 describes the events of this example and the equations used to create the corresponding MCR vectors. Notice that these events have a structure similar to the one described in Figure 2. The vectors denoted by **S** are reduced descriptions of the events, while the other vectors are the components of the full representation. Notice also that some of these component vectors (e.g. **mark**) are themselves reduced description vectors.

These equations in Table 2 are just one set out of many available options. For example, binding each event type vector (such as **eat**) with an event type role vector (e.g. $\mathbf{event}_{type}$) will facilitate the decoding of the event type. Table 3 lists the distances between the vectors that represent the events S_1 to S_6. The equations used to construct these vectors influence their similarity to each other. For example, S_4, S_5, and S_6 have short distances between each other, reflecting their similarity. S_6 is farther from S_5 than S_4 even though S_5 and S_6 share the same components; the difference in roles accounts for this. Including the agent and object fillers as extra terms in the equation increases the similarity between events with the same components even if they participate in different roles. For example, the definition of S_5 would change to:

$$\mathbf{S_5} = \mathbf{see} + \mathbf{see}_{\boldsymbol{agt}} \otimes \mathbf{john} + \mathbf{see}_{\boldsymbol{obj}} \otimes \mathbf{thefish} + \mathbf{john} + \mathbf{thefish} \qquad (52)$$

Table 3 here

Notice that the vectors S_1 and S_2 defined in Table 2 have no similarity (see Table 3), even though they have some common elements. This is because of the way we coded these two different types of ideas (a statement and a cause-effect). Using definitions similar to equation (52) for them, which include the components as terms, they would have some similarity and their distance would be shorter.

The decoding using probing works as follows: multiplying the event vector by the inverse of a role produces a vector similar to the filler vector, or in other words, the filler vector plus a small amount of noise. An auto-associative memory that contains all the vectors of the system works as a cleanup memory, which returns the closest vector to the one produced by the decoding. Table 4 shows several expressions that use this process. To better illustrate it, we will analyze the first expression in this table in detail.

$$\boldsymbol{E_1} = \mathbf{S_1} \otimes \mathbf{eat}_{\boldsymbol{agt}}{}^{\mathbf{-1}} = \left(\mathbf{eat} + \mathbf{eat}_{\boldsymbol{agt}} \otimes \mathbf{mark} + \mathbf{eat}_{\boldsymbol{obj}} \otimes \mathbf{thefish}\right) \otimes \mathbf{eat}_{\boldsymbol{agt}}{}^{\mathbf{-1}} \tag{53}$$

Since the multiplication is distributive over the sum, we can rewrite the expression as

$$\boldsymbol{E_1} = \mathbf{eat} \otimes \mathbf{eat}_{\boldsymbol{agt}}{}^{-1} + \mathbf{eat}_{\boldsymbol{agt}} \otimes \mathbf{mark} \otimes \mathbf{eat}_{\boldsymbol{agt}}{}^{-1} + \mathbf{eat}_{\boldsymbol{obj}} \otimes \mathbf{thefish} \otimes \mathbf{eat}_{\boldsymbol{agt}}{}^{-1} \tag{54}$$

According with the multiplication by the inverse, the second term produces the vector **mark**. Since the multiplication produces vectors dissimilar to it operands, the other two terms produce vectors that are dissimilar to any other vector that we have used in this experiment, and we can consider them to be noise. And finally, since the sum produces vectors similar to its operands, the expression is similar to the vector **mark**.

$$\boldsymbol{E_1} = \boldsymbol{noise} + \mathbf{mark} + \boldsymbol{noise} \approx \mathbf{mark} \tag{55}$$

As shown in Table 4, cueing an auto-associative cleanup memory (which contains the vectors that we defined in our experiments) with $\boldsymbol{E}_1$, retrieves the vector **mark**. The other vectors representing persons (**luke**, **john,** and **paul**) are closer than chance (the indifference distance is 2,048), but farther away than **mark** by about 7 *SD*.

$\boldsymbol{E}_1$ is also similar to the two noise terms in the expression, but, as we have proven, using high dimensional vectors it is almost impossible that these vectors were close to any of the vectors in the experiment just by chance. On the other hand, if we have too many vectors similar to **mark**, there is a possibility that we would retrieve one of the other vectors from the cleanup memory. Plate [7] analyzed the dependency of this operation with the number of similar vectors stored in the cleanup memory for HRR vectors. An almost similar analysis can be derived for MCR vectors, but it is outside to the scope of this paper. Nevertheless, since MCR has better variance in the results of the operations than HRR (see details in Section 8), it is more noise robust, and as a consequence, would support more similar vectors in the clean up memory than HRR, and still produce the correct result.

Table 4 here

This example also demonstrates the difference between the *chunking* mechanism and *holistic* processing [7]. Chunking involves a sequence of operations. For example, the expression in line 5 can decode S_1, the object of S_2, which is itself a composite vector. By first cleaning up the vector S_1, and then applying the expression in line 1, we obtain **mark**, the agent of S_1. On the other hand, using holistic processing produces the same result in one operation, as showed by the expression in line 6, which yields the final result directly without decoding the intermediate vector S_1. Chunking produces less noise than holistic processing, but requires an extra cleanup operation.

Also interestingly, the expression in line 8 returns random vectors, which are almost at the indifference distance from any vector used in the system, because S_3 does not have an object component, and the expressions of lines 10 and 11 that correctly decode John's role in similar events.

Notice that the base and id vectors that we have chosen randomly define the whole set of vectors used in the experiment. Choosing another collection of random vectors for the base and id vectors will lead to a different set of values for all the vectors in this experiment, but the overall result would be almost identical. We ran this same experiment 100 times with different base vectors in each run, and in each of them we got results consistent with the ones reported here.

MCR can employ Integer SDM as cleanup memory. Performing this same experiment using Integer SDM with a word length of 512 dimensions, 100,000 hard locations and a radius of activation of 1,925 (see [26, 27]) produces results similar to those reported above, with a few notable considerations. Some of the expressions in Table 4, in particular lines 2, 6, and 7, return vectors with an elevated level of noise compared to the target vector, producing retrieval errors in a few of the runs. Increasing the radius of activation of the hard locations in the memory mitigates this problem. The rest of the expressions yield vectors that retrieve the correct values in all the trials. To simulate extra data, 1,000 random vectors were preloaded in the memory. Notice that the retrieval errors mentioned here result from Integer SDM's limitations as a cleanup memory, and not from those of the MCR model. Employing a different cleanup memory, such as the simple hash table and exhaustive search procedure described earlier, does not produce these errors in this experiment. Nevertheless, the limitation of MCR, and other reduced description models, as pointed out by Plate [7], is given by the number of similar vectors (e.g. vectors representing people) stored in the clean up memory. A detailed analysis of this effect and possible ways to address this limitation are outside of the scope of the present paper; however they are topics for a future work.

MCR can model other data structures, representations, and applications as described in [7, 8]. By adapting the procedures presented in [25], MCR can represent sequences and related structures efficiently. Moreover, the use of random permutations is completely compatible with MCR, which allows employing them as an alternative to the multiplication described here. Using MCR, it is possible to reproduce all the experiments described by Plate [7] and Kanerva [8]. Among other examples, Kanerva describes the substitution of one attribute (e.g. color red) for another (e.g. color green) in a reduced description vector without first extracting the full representation. He also presents an analogy representation using coin denominations and countries, and shows how to use these representations to resolve logic problems. Plate describes several experiments with analogies and comparisons with figures and sentences that produce results similar to their psychological counterparts. We have already reproduced some of these same experiments using MCR vectors with results similar to the ones reported by Plate and Kanerva [28]. Since these experiments do not contribute to the current discussion, additional repetition of experiments and further analysis of them are unnecessary.

7. Normalized Distance and Similarity

The distance defined for MCR has an inconvenient dependence on *n*, the dimensionality of the vectors, and *r*, the number of possible values, making it difficult to compare the performance of MCR models with different values for these parameters. A *normalized distance* independent of *r* and *n*, denoted by *d'*, becomes useful for these comparisons:

$$d'(A,B) = d(A,B)\frac{4}{nr} \tag{56}$$

Its distribution is approximately normal with the following mean and variance:

$$D' \sim N\left(1, \left(\frac{1}{3} + \frac{8}{3r^2}\right)\frac{1}{n}\right) \tag{57}$$

Figure 5 here

The minimum normalized distance is zero, as in the non-normalized distance, but using *d'* the value one corresponds to the indifference distance, and the value two to its maximum. The distribution of *D'* clearly shows that its variance diminishes proportionally with *n* without bound, allowing the creation of a model with a distance distribution variance as low as desired. Notice that a model with a small variance has high noise robustness, accuracy, and reliability. The variance also diminishes when incrementing *r*; however, when *r* becomes large, the second term in the sum tends to zero, and 1/3 dominates. If *r* is 16 or greater, the value of the variance tends to the maximum possible (for a given value of *n*). The worst value corresponds to r equal to two, the binary case. See Figure 5 for details.

The *similarity* among vectors, defined as

$$sim(A,B) = 1 - d'(A,B) \tag{58}$$

is particularly handy for comparing results to those of models that use other similarity measures, such as the cosine. A vector has a similarity of one with itself, and zero similarity when compared with vectors at the indifference distance (corresponding to a normalized distance of one). The distribution of similarities of one vector with all the other vectors in the space is almost the same as *D'*, but with a mean equal to zero:

$$Sim \sim N\left(0, \left(\frac{1}{3} + \frac{8}{3r^2}\right)\frac{1}{n}\right) \tag{59}$$

8. Expected Value and Variance of the Similarity of Selected Expressions

Plate [7] discussed the means and variances of different similarity measures among several prototypical expressions of HRR in the frequency domain. Here we compare Plate's results with our calculations using MCR. Our experiments employ 512-dimensional vectors, matching the configuration used by Plate, and *r* =16.

Table 5 shows the theoretical values and the experimental results using MCR for several expressions that were also described by Plate using HRR [7]. Notice that the operations in the expressions are deterministic. In other words, with the same vectors *A*, *B*, *C*, and *D*, the expressions always produce the same results. The means and variances in the table compare the analytical estimates and experimental results after calculating each expression multiple times with different random vectors.

Due to the properties of the multiplication described previously, multiplying a vector *A* by another vector *B,* and then by its inverse B^{-1} yields exactly the same vector *A*, which explains the theoretical results of the expressions with mean one and variance zero. The rest of the expressions in the table compute the similarity between unrelated vectors, thus they follow the distribution of equation (59) with

r = 16, and a mean equal to zero and variance normalized by n equal to $\frac{1}{3}+\frac{8}{3r^2}=0.34375$. The experimental values in the table show the results of 50,000 runs for each expression, all of which have negligible differences when compared to the analytical results. Notice that the variance is normalized by n, which means that the actual variance is n times smaller. For example, for 1,000 dimension vectors, the variance is 0.34375×10^{-3}.

Table 5 here

HRR in the frequency domain [7] has the same means for each of these expressions, but with higher variances, 0.5 compared to 0.34375 in MCR. When r = 2, MCR is equivalent to Spatter Code, which has a variance of one for these same expressions. In conclusion, MCR is more noise robust than either HRR or Spatter Code for models using vectors with the same size n. Notice that in the limit as r approaches infinity, the normalized variance of the similarity of vectors in MCR tends to 1/3; the value for r = 16 is not far from this theoretical minimum, and values greater than 16 do not significantly improve the normalized variance. In consequence, r = 16 is a good choice for constructing MCR vectors, enabling the representation of such values with only 4 bits per dimension, and also limiting the storage requirements of Integer SDM memories, which increases linearly with r (see [26, 27] for details).

For the grouping operation, the analytical calculation of the means and variances are harder to obtain. Here we present the analysis for grouping two vectors, and measure the similarity to one of the operands. We also present the experimental results for grouping 2 to 15 vectors.

To analyze the mean and variance of

$$sim(A, A+B) \tag{60}$$

we can rewrite (60) as

$$sim(A, A+B) = 1 - d'(A, A+B) \tag{61}$$

According to the definition of the sum, the output of grouping two vectors has a distance to any of the operands equal to half of the distance between them.

$$sim(A, A+B) = 1 - \frac{d'(A,B)}{2} \tag{62}$$

Given that *d'* approximately follows the normal distribution in equation (57), $sim(A, A + B)$ also distributes normally:

$$Sim(A, A + B) \sim N\left(\frac{1}{2}, \frac{1}{4}\left(\frac{1}{3} + \frac{8}{3r^2}\right)\frac{1}{n}\right) \tag{63}$$

The normalized variance of the similarity given by grouping two vectors and one of its operands is approximately 0.086 for $r = 16$, and 0.084 for $r = 32$. The mean reported by Plate [7] using HRR is higher (0.6366). However, due to the difference in the variance of the distance distributions of the two spaces (remember that HRR uses cosine as its similarity measure), both models have almost the same probability of presenting the mean or less similarity between the sum vector and one of its operands just by chance. In other words, the cumulative density function (cdf) for the distributions of the similarity measure for HRR and MCR, have almost the same value for similarity, equal to 0.6366 and 0.5 respectively:

$$\mathrm{cdf}(Sim_{HRR}(A, A + B) = 0.6366) \cong \mathrm{cdf}(Sim_{MCR}(A, A + B) = 0.5) \tag{64}$$

Figure 6 here

Furthermore, the normalized variance of this similarity using MCR is smaller than in HRR: 0.086 as compared to 0.0947, which makes MCR more noise robust and accurate compared to HRR for a given dimensionality.

Figure 6 shows the experimental results of the similarity between a random vector and the same vector grouped with $k – 1$ other random vectors, for values of k between one and fifteen. The experiments use vectors with 512 dimensions and r with values 2, 16, and 32. The data correspond to 10,000 runs for each value of k. The results for $k = 2$ confirm the theoretical analysis. Additionally, compared with HRR, MCR has better variances and similar means, considering the cdf of the distributions of the similarity functions, also making it less noisy for grouping. See [7] for details.

9. Summary of Comparisons: MCR, HRR and Spatter Code

MCR shares many properties with HRR and Spatter Code. All three enable reduced descriptions. Actually, MCR is a generalization of Spatter Code that uses modular integer vectors instead of binary vectors. When $r = 2$, MCR becomes equivalent to Spatter Code. Our analytical and experimental results show that MCR is more reliable and accurate than Spatter Code for a given number of dimensions; however, Spatter Code utilizes simpler operations, which would be an advantage for some applications.

The representational expressiveness of MCR would be considered a further advantage over Spatter Code in applications that require the encoding of non-binary data. (See for example [14].) Finally, Spatter Code employs XOR as its binding operation. This operation has the drawback that the inverse vector is the same vector (See equation (2)). This prevents multiplying by the same vector repeated times, which is useful, for example, for representing sequences. See details in [7, 8, 25]. MCR does not have this problem.

Although HRR has a rich expressive representation and very good performance when combining and decoding structures from holistic vectors, it utilizes complex operations, such as circular convolution, that have time complexities of the order of $O(n^2)$ or $O(n\text{Log } n)$. HRR in the frequency domain, also known as circular HRR, has better overall performance, can perform the operations in $O(n)$ time, and has more stable variances and results than the normal HRR. As Kanerva pointed out in a personal communication with the authors, under an interpretation of the values in MCR as discretized angles, the binding and grouping operations of both models are similar. However, each model utilizes a different distance (or similarity) measurement, which explains the variations in performance between the two models. MCR was conceived as an extension of Spatter Code, inheriting the simplicity of the latter's design. On the other hand, circular HRR was derived from normal HRR, producing a more cumbersome base for the model. Finally, MCR can readily utilize Integer SDM as its cleanup memory, whereas HRR has no specific auto-associative memory available.

10. Conclusions and further work

MCR is a new reduced description representation that balances representational expressiveness with implementational simplicity. It has all the required and desirable characteristics of reduced descriptions: representation adequacy, reduction, systematicity, and informativeness. Moreover, it implements explicit and structural similarity (i.e. the similarity of the structure, as opposed as the content, of a hierarchy or sentence), which allows the holistic processing of several operations, avoiding the need to reconstruct the structure prior to processing.

The experiments and analysis detailed herein have demonstrated MCR's performance in a number of scenarios, empirically validating its anticipated noise robustness, representational expressiveness, and holistic processing capability. The analysis of the means and variances for the similarities of representative operations suggests that MCR has better performance for these operations than HRR or Spatter Code using vectors with the same number of dimensions. Nevertheless, the accuracy of any of these models can be increased without bound by enlarging the dimensionality of the vectors.

To perform the experiments, we developed a script parser and interpreter that allows writing the expressions and operations of MCR in a simple language, and running it embedded within a Java program. This greatly facilitates the creation and running of experiments and applications that use MCR.

Certain MCR characteristics correlate well with those of biologically plausible models. The high dimensional vectors might represent the activation of distributed units, such as neurons. Other vector models of the so called *deep learning* systems, such as HMAX [29], HTM [30], DeSTIN [31], and deep belief nets [32, 33] are based on the hierarchical organization of the neocortex, and particularly, of the visual cortex. They focus on learning and recognition of spatial and temporal patterns. An interesting path of future research would be to determine if the vectors produced by these systems might be combined using MCR to form more complex representations.

Several *challenging* AI applications, such as cognitive architectures, natural language understanding, or visual recognition, might benefit from MCR. Some of the characteristics of this vector representation–noise robustness, explicit similarity, and structural similarity–can facilitate the implementation of such applications. The simplicity of this model's operations, and its performance make it an attractive option over other models. For example, the current implementation of the LIDA cognitive architecture [3, 34] employs nodes and links in a graph-like structure as its main representation model. For representing an object, it uses a node for the object itself, and other nodes that represent its features (e.g. red, big, etc). Each link that connects the object node with one of the feature nodes has also a specific associated meaning (e.g. color, size, etc.). In more complex examples, the graph of nodes and links may be large and complex. Comparing these structures is a very common operation in this cognitive architecture, but it is hard to perform. Performing *approximated* comparisons, which is also required in this architecture, is even harder using this representation. On the other hand, using MCR vectors, approximated comparisons of structures is an intrinsic property of the representation.

Sequence representation and learning are also examples of common operations in many of these challenging applications, in particular in cognitive architectures. Several authors, including Starzyk and He [35] and Sun & Giles [36], consider spatial and temporal sequence learning to be one of the most important forms of learning: sequences in procedural learning, learning new skills, high level planning, and problem solving. MCR vectors can represent sequences, and allow comparing sequences, even if they are not identical (approximate comparison). See [25] for an example of sequences represented using reduced description vectors.

A promising project, Vector LIDA, would implement the LIDA cognitive architecture using MCR vectors as its main representation for data structures. Some of the advantages over the current implementation, which employs nodes and links in a graph-like structure, includes a more realistic and biologically plausible model, better integration with the episodic memory, which already uses a vector based SDM memory, better integration with other low level perceptual processing (such as HMAX [29]), better scalability, and easier learning mechanisms.

11. Acknowledgments

We want to thank Steve Strain for his suggestions and comments, and for his help in the editing of this manuscript. We are also indebted to several anonymous reviewers whose comments helped us to greatly improve the paper.

References

1. Hinton GE, McClelland JL, Rumelhart DE. Distributed representations. In: Rumelhart DE, McClelland JL, editors. Parallel Distributed Processing: Explorations in the Microstructure of Cognition Volume 1: Foundations. Cambridge, MA: MIT Press; 1986. p. 77-109.
2. Deerwester SC, Dumais ST, Furnas GW, Landauer TK, Harshman RA. Indexing by latent semantic analysis. Journal of the American Society of Information Science. 1990;41(6):391-407.
3. Franklin S, Patterson FGJ. The LIDA Architecture: Adding New Modes of Learning to an Intelligent, Autonomous, Software Agent. IDPT-2006 Proceedings (Integrated Design and Process Technology): Society for Design and Process Science; 2006.
4. Laird JE. Extending the Soar Cognitive Architecture. In: Wang P, Goertzel B, Franklin S, editors. Artificial General Intelligence 2008. Amsterdam: IOS Press.; 2008.
5. Hinton GE. Mapping part-whole hierarchies into connectionist networks. Artificial Intelligence. 1990;(46):47-75.
6. Plate TA. Holographic Reduced Representations. IEEE Transactions on Neural Networks. 1995;6(3):623-41.
7. Plate TA. Holographic Reduced Representation: distributed representation of cognitive structure. Stanford: CSLI; 2003.
8. Kanerva P. Hyperdimensional Computing: An Introduction to computing in distributed representation with high-dimensional random vectors. Cognitive Computation. 2009;1(2):139-59.
9. Stewart TC, Eliasmith C, editors. Neural Planning and Reasoning Using the Synaptic Connections of the Basal Ganglia and Thalamus. Biologically Inspired Cognitive Architectures 2011; 2011; Whashington, DC: IOS Press.
10. Franklin S. Artificial Minds. Cambridge MA: MIT Press; 1995.
11. Kanerva P. Sparse Distributed Memory. Cambridge MA: The MIT Press; 1988.
12. Winston PH. Artificial Intelligence. 3rd ed. Boston, MA: Addison Wesley; 1992.
13. Foundalis HE. PHAEACO: A Cognitive Architecture Inspired by Bongard's Problems. PhD Thesis. Indiana: Indiana University; 2006.
14. Jockel S. Crossmodal Learning and Prediction of Autobiographical Episodic Experiences using a Sparse Distributed Memory. PhD Thesis. Hamburg: University of Hamburg; 2009.
15. Robertson P, Laddaga R, editors. Learning to Find Structure in a Complex World. Biological Inspired Cognitive Architectures 2011; 2011; Washington DC.

16. Sahlgren M, editor. An Introduction to Random Indexing. Methods and Applications of Semantic Indexing Workshop at the 7th International Conference on Terminology and Knowledge Engineering, TKE 2005; 2005; Copenhagen, Denmark.
17. Cohen T, Widdows D. Empirical distributional semantics: Methods and biomedical applications. Journal of Biomedical Informatics. 2009;42(2):390-405.
18. Turney PD, Pantel P. From Frequency to Meaning: Vector Space Models of Semantics. Journal of Artificial Intelligence Research. 2010;37:141-88.
19. Cambria E, Hussain A. Sentic Computing. Hussain A, editor: Springer; 2012.
20. Jones MN, Mewhort DJK. Representing word meaning and order information in a composite holographic lexicon. Psychological Review. 2007;114:1-37.
21. Kanerva P. The binary spatter code for encoding concepts at many levels. In: Marinaro M, Morasso P, editors. ICANN '94: Proceedings of International Conference on Artificial Neural Networks. London, UK: Springer-Verlag; 1994. p. 226–9.
22. Pollack JB. Recursive Distributed Representations. Artificial Intelligence. 1990;46(1-2):77-105.
23. Willshaw DJ, Buneman OP, Longuet-Higgins HC. Non-holographic associative memory. Nature. 1969;222(5197):960-2.
24. Rachkovskij DA, Kussul EM. Binding and Normalization of Binary Sparse Distributed Representations by Context-Dependent Thinning. Neural Computation. 2001;13(2):411-52.
25. Snaider J, Franklin S. Extended Sparse Distributed Memory and Sequence Storage. Cognitive Computation. 2012;4(2):172-80.
26. Snaider J, Franklin S, Strain S, George EO. Integer Sparse Distributed Memory: Analysis and Results. Neural Networks. 2013;46:144-53.
27. Snaider J, Franklin S, editors. Integer Sparse Distributed Memory. The 25th Florida Artificial Intelligence Research Society Conference FLAIRS-25; 2012; Marco Island, FL.
28. Snaider J. Integer Sparse Distributed Memory and Modular Composite Representation. PhD Thesis. Memphis, TN: University of Memphis; 2012.
29. Serre T, Wolf L, Bileschi S, Riesenhuber M, Poggio T. Robust Object Recognition with Cortex-Like Mechanisms. IEEE Transations on Pattern Analysis and Machine Intelligence. 2007;29(3):411-26.
30. George D. How the brain might work: A hierarchical and temporal model for learning and recognition. PhD Thesis.: Stanford University; 2008.
31. Arel I, Rose D, Coop R. DeSTIN: A Scalable Deep Learning Architecture with Application to High-Dimensional Robust Pattern Recognition. Proc of the AAAI 2009 Fall Symposium on Biologically Inspired Cognitive Architectures (BICA)2009.
32. Hinton GE. Learning multiple layers of representation. TRENDS in Cognitive Sciences. 2007;11(10):428-34.
33. Hinton GE, Osindero S, Teh Y. A fast learning algorithm for deep belief nets. Neural Computation. 2006;18:1527-54.
34. Snaider J, McCall R, Franklin S. The LIDA Framework as a General Tool for AGI. The Fourth Conference on Artificial General Intelligence; Mountain View, CA2011.
35. Starzyk JA, He H. Anticipation-Based Temporal Sequences Learning in Hierarchical Structure. IEEE Transactions on Neural Networks. 2007;18(2):344-58.
36. Sun R, Giles CL. Sequence learning: From recognition and prediction to sequential decision making. IEEE Intelligent Systems. 2001;16(4):67-70.

Table 1. Distances among some vectors of the example. The diagonal, with distances equal to 0, corresponds to the distance of a vector with itself. Notice that vectors with common features, such as the vectors that represent persons, are close (see text for a definition of "close").

	mark	john	paul	luke	thefish	thebread	hunger	thirst
mark	0							
john	1078	0						
paul	1101	1113	0					
luke	1121	1125	1088	0				
thefish	2008	1978	2027	1965	0			
thebread	2102	2084	2099	2077	1502	0		
hunger	2033	2027	2044	2046	2033	2009	0	
thirst	2036	2012	1995	1975	2068	2034	1345	0

Table 2. Events created using the token and role vectors of the example.

Event	Equation
Mark ate the fish.	$\mathbf{S_1} = \mathbf{eat} + \mathbf{eat}_{\boldsymbol{agt}} \otimes \mathbf{mark} + \mathbf{eat}_{\boldsymbol{obj}} \otimes \mathbf{thefish}$
Hunger caused Mark to eat the fish.	$\mathbf{S_2} = \mathbf{cause} + \mathbf{cause}_{\boldsymbol{agt}} \otimes \mathbf{hunger} + \mathbf{eat}_{\boldsymbol{obj}} \otimes \mathbf{S_1}$
John ate.	$\mathbf{S_3} = \mathbf{eat} + \mathbf{eat}_{\boldsymbol{agt}} \otimes \mathbf{john}$
John saw Mark.	$\mathbf{S_4} = \mathbf{see} + \mathbf{see}_{\boldsymbol{agt}} \otimes \mathbf{john} + \mathbf{see}_{\boldsymbol{obj}} \otimes \mathbf{mark}$
John saw the fish.	$\mathbf{S_5} = \mathbf{see} + \mathbf{see}_{\boldsymbol{agt}} \otimes \mathbf{john} + \mathbf{see}_{\boldsymbol{obj}} \otimes \mathbf{thefish}$
The fish saw John.	$\mathbf{S_6} = \mathbf{see} + \mathbf{see}_{\boldsymbol{agt}} \otimes \mathbf{thefish} + \mathbf{see}_{\boldsymbol{obj}} \otimes \mathbf{john}$

Table 3. Distances among vectors representing the events described in Table 2.

	S_1	S_2	S_3	S_4	S_5	S_6
S_1	0					
S_2	1947	0				
S_3	1159	2002	0			
S_4	1995	2036	1830	0		
S_5	1858	1983	1839	1085	0	
S_6	2025	2024	2036	1390	1443	0

Table 4. Results of unbinding components from event vectors.

Description	Expression	Rank of distances			
1. Agent of eating of S_1	$\boldsymbol{E_1} = \mathbf{S_1} \otimes \mathbf{eat}_{\boldsymbol{agt}}{}^{\mathbf{-1}}$	**mark** (1181)	**luke** (1491)	**paul** (1523)	**john** (1593)
2. Agent of S_1	$\boldsymbol{E_2} = \mathbf{S_1} \otimes \mathbf{agent}^{\mathbf{-1}}$	**mark** (1554)	**luke** (1652)	**paul** (1660)	**john** (1778)
3. Object of S_1	$\boldsymbol{E_3} = \mathbf{S_1} \otimes \mathbf{eat}_{\boldsymbol{obj}}{}^{\mathbf{-1}}$	**thefish** (1166)	**food** (1629)	**fish** (1666)	**thebread** (1837)
4. Agent of S_2	$\boldsymbol{E_4} = \mathbf{S_2} \otimes \mathbf{cause}_{\boldsymbol{agt}}{}^{\mathbf{-1}}$	**hunger** (1187)	**state** (1572)	**thirst** (1737)	**human** (1897)
5. Object of S_2	$\boldsymbol{E_5} = \mathbf{S_2} \otimes \mathbf{cause}_{\boldsymbol{obj}}{}^{\mathbf{-1}}$	$\mathbf{S_1}$ (1209)	**eat** (1620)	$\mathbf{S_3}$ (1628)	$\mathbf{S_5}$ (1908)
6. Agent of object of S_2	$\boldsymbol{E_6} = \mathbf{S_2} \otimes \mathbf{cause}_{\boldsymbol{obj}}{}^{\mathbf{-1}} \otimes \mathbf{eat}_{\boldsymbol{agt}}{}^{\mathbf{-1}}$	**mark** (1666)	**luke** (1804)	**paul** (1806)	**john** (1866)
7. Object of object of S_2	$\boldsymbol{E_7} = \mathbf{S_2} \otimes \mathbf{cause}_{\boldsymbol{obj}}{}^{\mathbf{-1}} \otimes \mathbf{eat}_{\boldsymbol{obj}}{}^{\mathbf{-1}}$	**thefish** (1659)	**food** (1886)	**fish** (1887)	$\mathbf{eat}_{agt}$ (1939)
8. Object of S_3	$\boldsymbol{E_8} = \mathbf{S_3} \otimes \mathbf{eat}_{\boldsymbol{obj}}{}^{\mathbf{-1}}$	**see** (1927)	$\mathbf{see}_{agt}$ (1947)	$\mathbf{S_6}$ (1959)	**state** (1966)
9. John's role in S_4	$\boldsymbol{E_9} = \mathbf{S_4} \otimes \mathbf{john}^{\mathbf{-1}}$	$\mathbf{see}_{agt}$ (1124)	**agent** (1459)	$\mathbf{eat}_{agt}$ (1634)	$\mathbf{see}_{obj}$ (1640)
10. John's role in S_5	$\boldsymbol{E_{10}} = \mathbf{S_5} \otimes \mathbf{john}^{\mathbf{-1}}$	$\mathbf{see}_{agt}$ (1120)	**agent** (1497)	$\mathbf{eat}_{agt}$ (1664)	$\mathbf{cause}_{agt}$ (1724)
11. John's role in S_6	$\boldsymbol{E_{11}} = \mathbf{S_6} \otimes \mathbf{john}^{\mathbf{-1}}$	$\mathbf{see}_{obj}$ (1129)	**object** (1527)	$\mathbf{eat}_{obj}$ (1637)	$\mathbf{cause}_{obj}$ (1715)

Table 5. Means and variances of selected expressions for a MCR model with n = 512 and r = 16. The experimental results correspond to 50,000 runs. The variance is normalized by multiplying by n.

Expression	Similarity			
	Analytic		Experimental	
	mean	n . var	mean	n . var
$sim(A, A)$	1	0.00000	1.0000	0.0000
$sim(A, B)$	0	0.34375	0.0000	0.3435
$sim(A, A \otimes B)$	0	0.34375	0.0000	0.3429
$sim(A, A \otimes C)$	0	0.34375	0.0000	0.3466
$sim(A, A \otimes A \otimes A^{-1})$	1	0.00000	1.0000	0.0000
$sim(B, A \otimes B \otimes A^{-1})$	1	0.00000	1.0000	0.0000
$sim(A, A \otimes B \otimes A^{-1})$	0	0.34375	0.0000	0.3430
$sim(C, A \otimes B \otimes A^{-1})$	0	0.34375	-0.0002	0.3412
$sim(C, A \otimes B \otimes C^{-1})$	0	0.34375	0.0001	0.3463
$sim(D, A \otimes B \otimes C^{-1})$	0	0.34375	-0.0001	0.3428

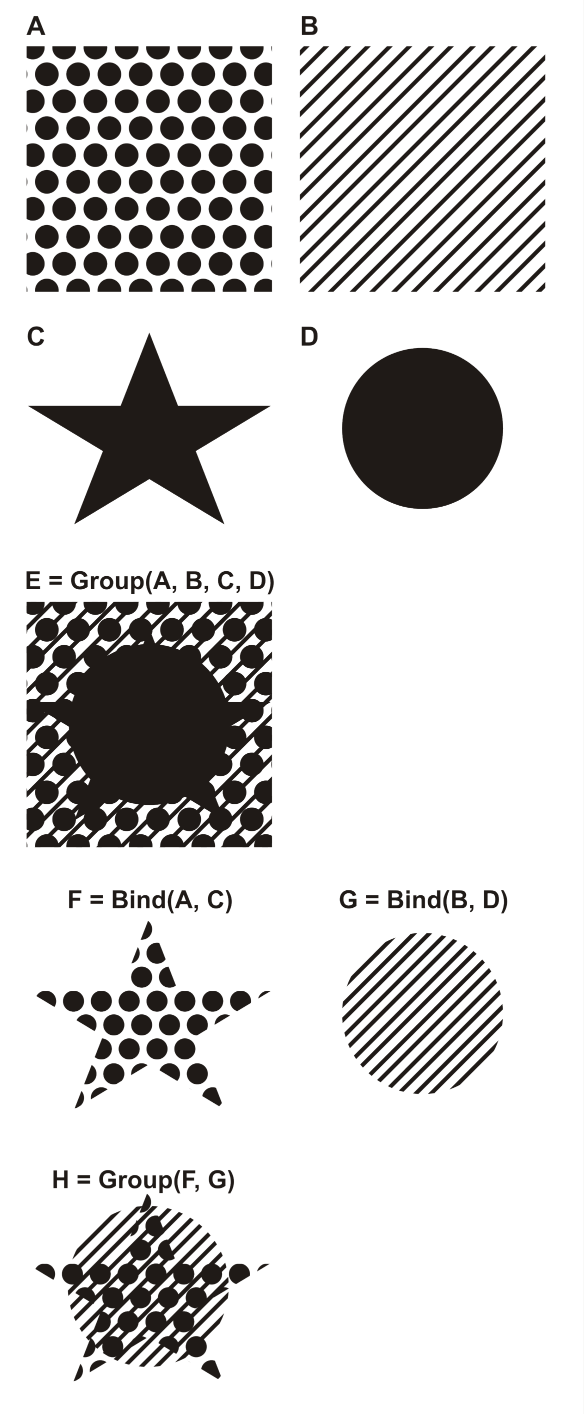

Figure 1. Grouping and Binding with images. Images A and B represent the features *first* and *second* respectively. For coding the statement *Star first and Circle second*, grouping alone is not enough (see image E). By binding A with C, and B with D (images F and G) first, and then grouping them, we obtain image H, which better represents the statement.

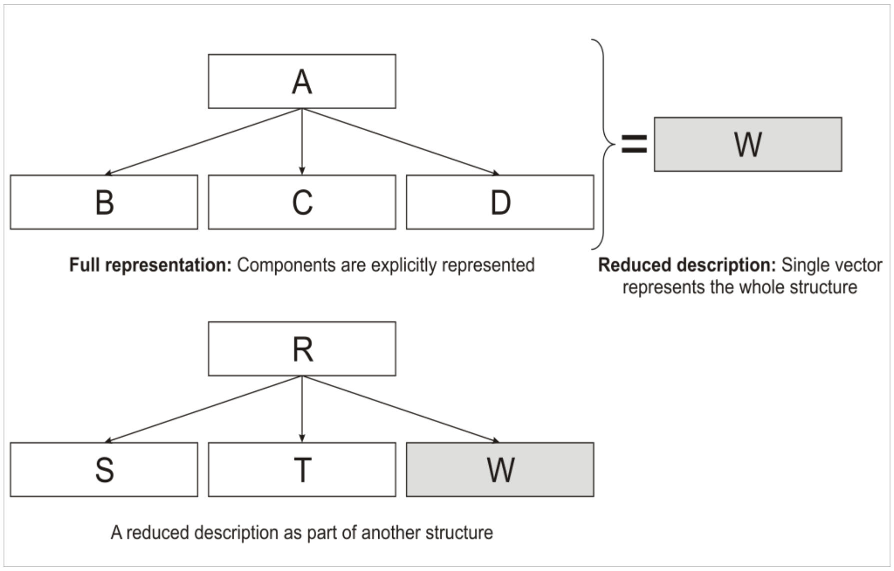
A
B
C
D
=
W
Full representation: Components are explicitly represented
Reduced description: Single vector represents the whole structure
R
S
T
W
A reduced description as part of another structure

Figure 2. Reduced description. A complex structure has a dual representation: a full representation with an explicit structure where each element is represented by a vector, and a reduced description, where a single vector represents the whole structure.

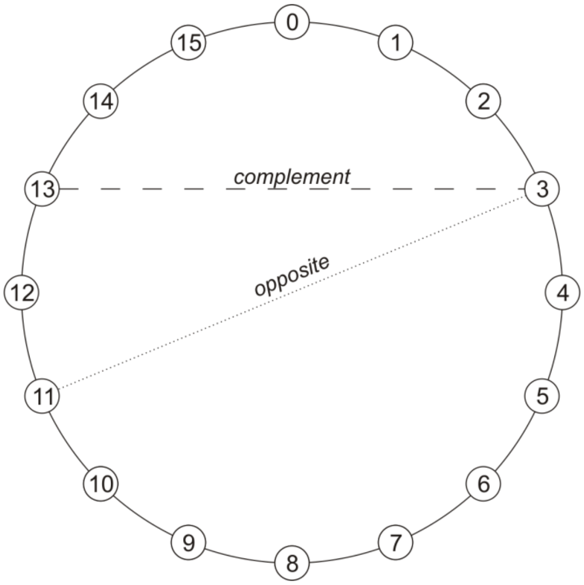

Figure 3. The possible values for one dimension of a modular integer vector with *r* = 16. The complement of a value is another value such that their sum equals *r*. The opposite of a value is the value in its *antipode*, that is, the value plus *r*/2.

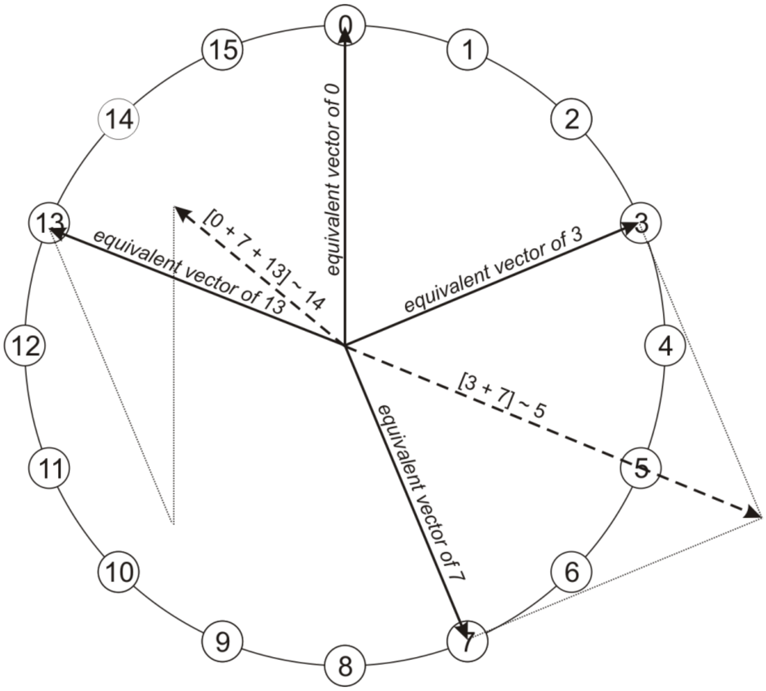

Figure 4. Equivalent vectors and examples of grouping.

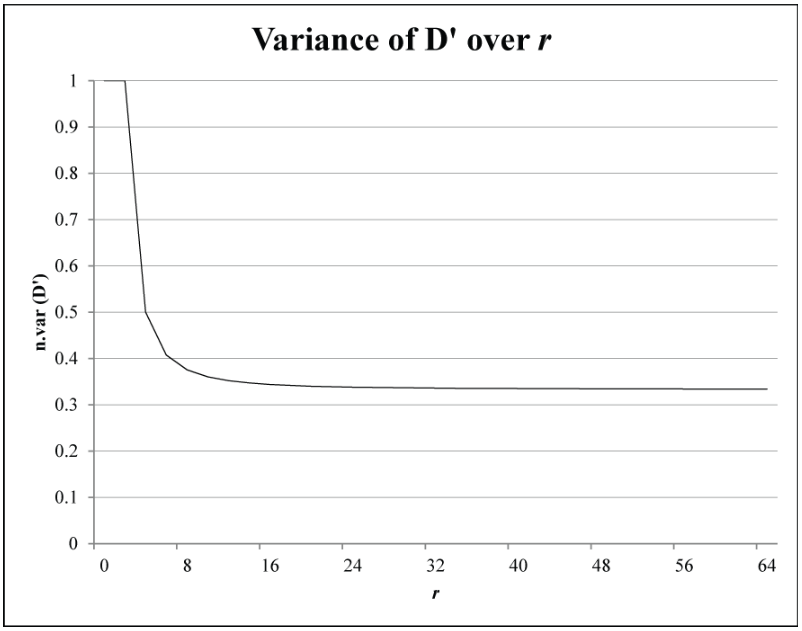

Figure 5. Variance of D’ over *r*.

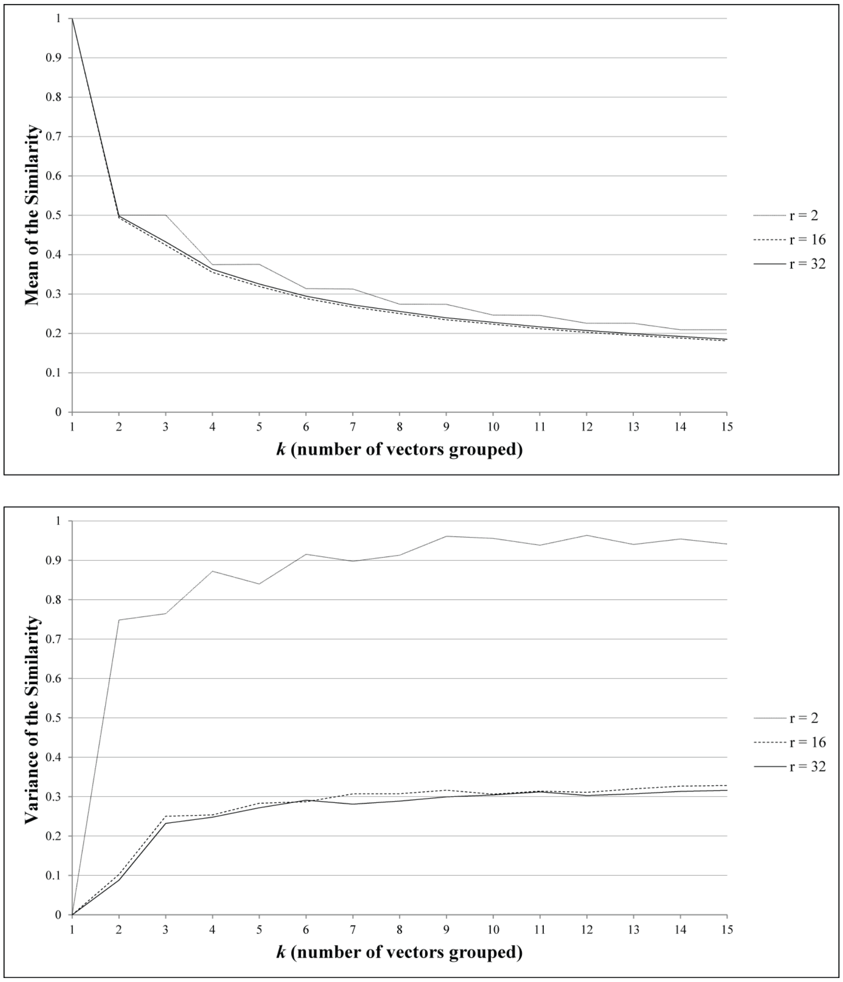

Figure 6. Means and variances of the similarity between a random vector and the same vector grouped with *k* – 1 other random vectors. The vectors have 512 dimensions and three values for *r* are evaluated (2, 16, and 32). The data correspond to 10,000 runs for each value of *k*.