GENERATING TENSOR REPRESENTATION FROM CONCEPT TREE IN MEANING BASED SEARCH

A Thesis

by

JAGANNATH PANIGRAHY

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2010

Major Subject: Computer Science

GENERATING TENSOR REPRESENTATION FROM CONCEPT TREE IN MEANING BASED SEARCH

A Thesis

by

JAGANNATH PANIGRAHY

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Rabi N. Mahapatra
Committee Members,	Eun Jung Kim
	Deepa Kundur
Head of Department,	Valerie E. Taylor

May 2010

Major Subject: Computer Science

ABSTRACT

Generating Tensor Representation from Concept Tree in

Meaning Based Search. (May 2010)

Jagannath Panigrahy, B.Tech.,

National Institute of Technology, Tiruchirapalli, India

Chair of Advisory Committee: Dr. Rabi N Mahapatra

Meaning based search retrieves objects from search index repository based on user's search Meanings and meaning of objects rather than keyword matching. It requires techniques to capture user's search Meanings and meanings of objects, transform them to a representation that can be stored and compared efficiently on computers. Meaning of objects can be adequately captured in terms of a hierarchical composition structure called *concept tree*. This thesis describes the design and development of an algorithm that transforms the hierarchical concept tree to a *tensor representation* using tensor algebra theory. These tensor representations can capture the information need of a user in a better way and can be used for similarity comparisons in meaning based search. A preliminary evaluation showed that the proposed framework outperforms the TF-IDF vector model in 95% of the cases and vector based conceptual search model in 92% of the cases in adequately comparing meaning of objects. The tensor conversion tool also was used to verify the salient properties of the meaning

comparison framework. The results show that the salient properties are consistent with the tensor similarity values of the meaning comparison framework.

To my parents

ACKNOWLEDGEMENTS

First I would like to sincerely thank my committee chair Dr. Rabi N Mahapatra for his invaluable guidance for this thesis. I would be forever grateful to him for his constant support and enormous patience during the course of my research. I would also like to thank my other committee members, Dr. Eun Jung Kim and Dr. Deepa Kundur for their guidance and support.

I would like to express my gratitude to all the professors who handled my courses during graduate studies for sharing their expertise with us. I would also like to thank all the department faculty and staff for their advice and help with all manner of administrative work.

I would also like to express my regards to Dr. Robert Coulson, and Dr. Andrew Birt for giving me an opportunity to work at the Knowledge Engineering Lab, Texas A&M University.

I would also like to thank my colleagues Amitava Biswas, Suneil Mohan and Aalap Tripathy, Suman Kalyan Mandal for their invaluable suggestions and help. I had a memorable time working with the embedded systems and co-design group.

Thanks also go to my friends Arupa Kumar Mohapatra, Srinath S, Shriram S, Harsha N, Sthiti Deka, Vijay Ramalingam, Anupam Jain, Kapil Garg, Vipin Kumar, Rahul Ravikumar, Ishan Desai and Jyotsna Priyadarshini for the wonderful time that we spent together at Texas A&M University.

Finally, thanks to my family for their constant encouragement, love and support.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

1. INTRODUCTION

In today's world internet has become the biggest resource for providing information on various topics. Search applications have become widespread and frequently used. A study has shown that an estimated 13 billion internet searches are being performed per month and it is growing at a rate of 38 percent annually [1]. With the increase in use of search applications there is also an increase in the expectations of users for better search performance. There is a continuous demand for meaning based search capabilities that can understand user query semantics. Today users look for a small set of precise results on a broad range of topics and they are more concerned with the precision of the search results compared to its recall [2]. Users also want search engines to provide different kinds of objects in query results like audio files, video files image files along with text files. The current Information retrieval technologies have several limitations to satisfy the current expectations of users. So there is a need for a new framework that can address the limitations of current Information Retrieval (IR) technologies. A meaning based search-framework [3, 4] can be a solution to address some of the limitations mentioned above.

The meaning based framework tries to address two key challenges of a meaning based search engine; they are Meaning Representation and Meaning Comparison. It describes a method that can be used to adequately capture the meaning of objects in

---

This thesis follows the style of *IEEE Journal of Solid State Circuits.*

terms of a hierarchical composition structure called *concept tree* [3, 4]. This thesis describes the design and development of an algorithm based on a *Tensor algebra* theory that converts a *concept tree* to a Tensor representation which can be stored and used for similarity comparison on computers. Two tensor representations can be used for similarity comparison by taking the inner dot product of the basis vectors which is analogous to the vector model approach. As part of the thesis work a Java based concept tree to tensor conversion application is developed using the proposed algorithm and used for simulations to carry out experiments and evaluate the Meaning based framework.

This thesis is organized as follows. In the next section, we present a brief introduction to the Tensor Algebra theory and the proposed algorithm to convert a concept tree to a tensor representation. We then follow up with a brief discussion on the salient properties of the proposed Tensor Algebra and simulations that verifies the properties. Then we conclude by discussing some implications of our results and possible future work ideas.

In this section, we first present a brief introduction to our meaning based search framework and provide description about concept trees. We then talk about the motivation behind the problem and summarize some of the related research work in this area. Finally, we include a section that enlists the notation used in the rest of the thesis.

1.1 Meaning Based Search

Before going into details of a meaning based search engine, let's explore the traditional search engine system. A search engine is a system that collects and organizes

content from all over the internet. Those wishing to locate something would enter a query about what they'd like to find and the engine provides links to content that matches the user need. An abstract view of the above idea can be thought of by imagining the internet as a collection that stores objects and each object is associated with a key that can uniquely identify the object in the collection as shown in Figure 1. The search system matches the user query against all the keys and retrieves the matched documents. To make this comparison efficient and fast, an index data structure is build which stores the Key-object mappings and can be used for fast and accurate information retrieval. [5]

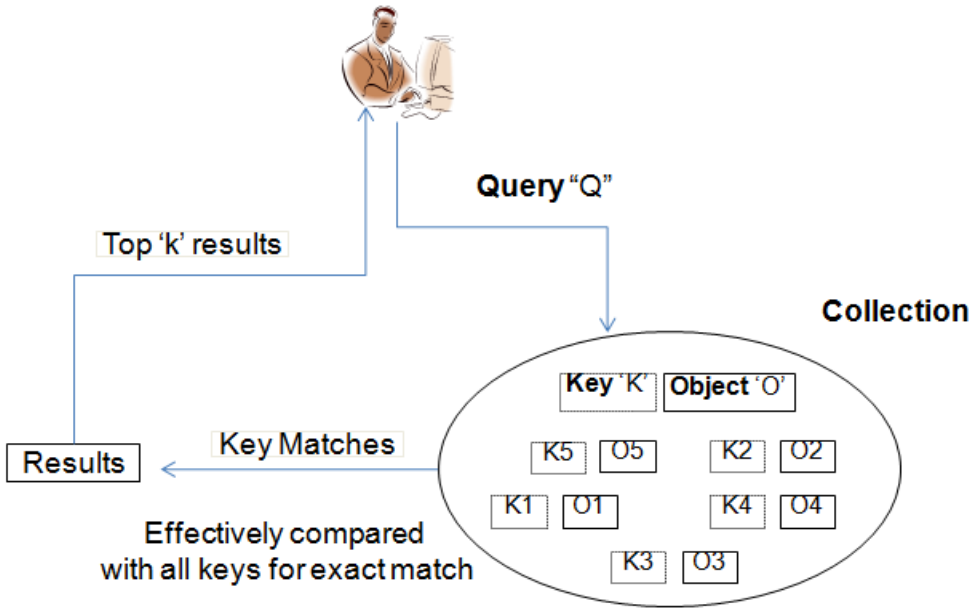

Figure1 Overview of search process

Meaning based search system is a system Search system that identify objects based on user's search intention and object's meaning rather than simple keyword matching.

Such a system can capture the user information need in a batter way and can give better results to end user with higher *precision* [2]. The meaning based search process can be viewed as below. Here we require some transformation techniques that can transform the object descriptions and user intentions to an appropriate *key* for comparisons as shown in Figure 2.

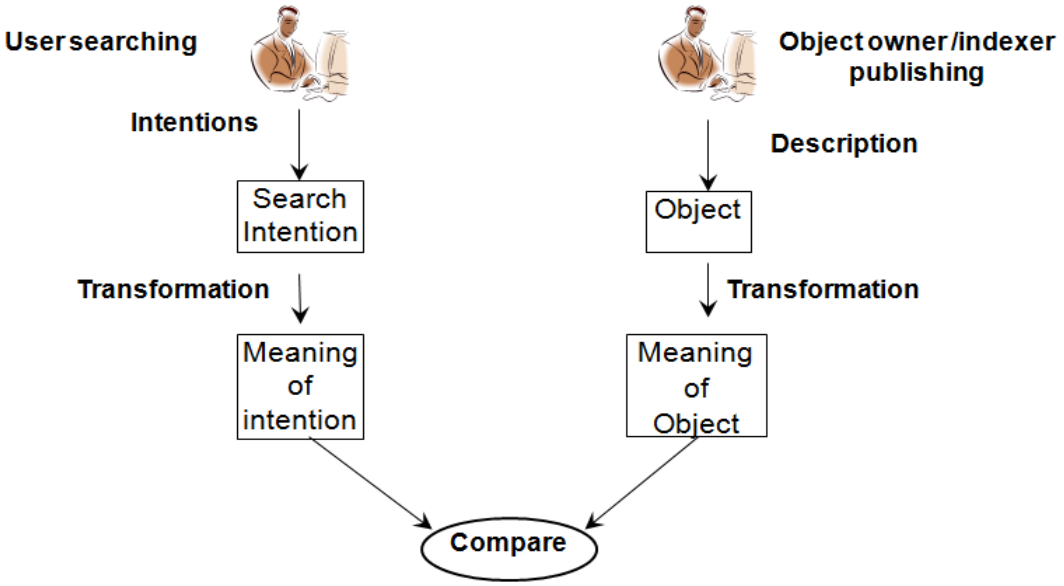

Figure 2 Transformations involved in meaning based search

There are many challenges involved in realizing a meaning based search system. Two of the key challenges are *Meaning Representation* and *Meaning Comparison*.

Two descriptions can have the same set of keywords but the meanings can be completely different [3, 4]. If keyword set will be used to represent the two descriptions then it is not a good technique for meaning based comparison. So there is a need for a new technique that can represent meaning appropriately. Adequate meaning representation determines search system efficiency in terms of accurate information retrieval.

We also require techniques that can compare the meaning representations efficiently as it is the core process of any information retrieval process that determines its efficiency in terms of computation speed.

Meaning based search framework proposes a new technique for meaning based searching. This framework describes a meaning based comparison model [5, 7]. The model proposes a technique to capture the meaning from textual descriptions of objects. It creates a semantic key called s*emantic descriptor* from the object description and builds a search index repository [1, 3, 4]. Similarly it constructs a search key *semantic descriptor* for the user query and uses it for searching. This technique allows users to successfully retrieve results based on object descriptions and not merely through keyword matching as done by most vector based search models. The generation of semantic descriptor is a multi-step process. The object description is captures by a hierarchical composition structure called a *Concept Tree* [3, 4]. The concept tree represents the complex meaning [8] of the user Meanings (or objects in index repository) though a hierarchical composition of concepts. The higher level complex concepts are represented in terms of a hierarchy of simpler concepts. Thus the leaf nodes of the

concept tree are elementary concepts in the domain ontology which requires no further decomposition. A detailed description of *concept tree* design and rationale behind it is given later in the section.

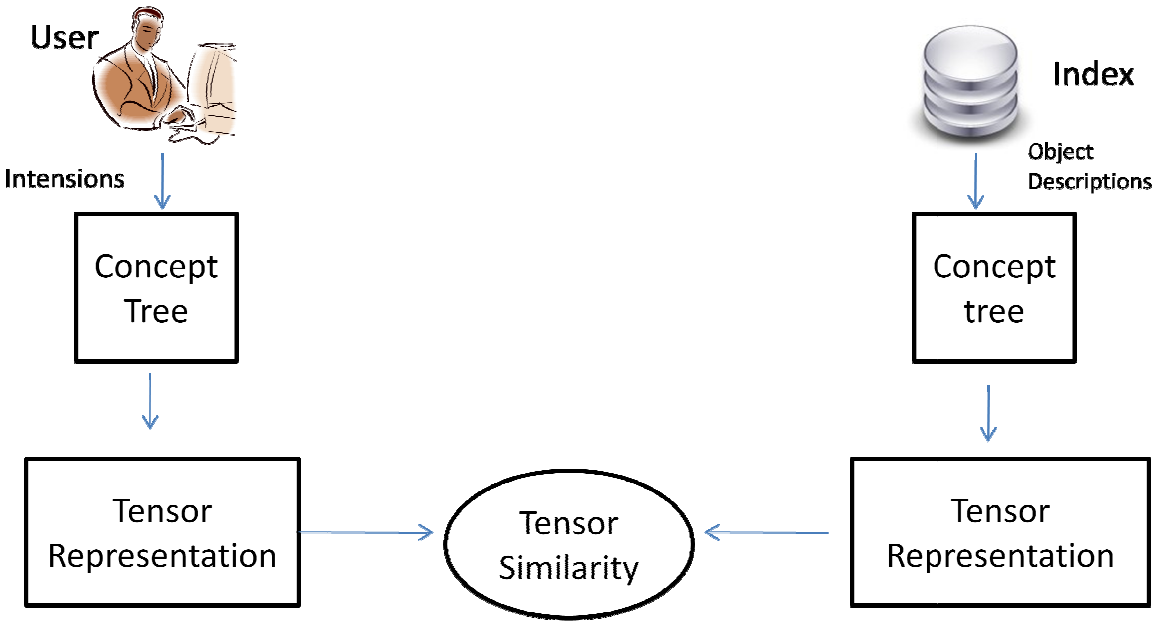

Figure 3 Meaning based search model

1.2. Motivation & Related Work

To realize the Meaning based search model as shown in Figure 3 we need a technique to store and compare the *concept trees* on computers to find out how similar they are. We need a meaning representation technique to successfully represent the *concept tree* structure on computers and a comparison algorithm to compare two concept trees. There are some meaning representation and tree comparison techniques proposed earlier. Some of the existing techniques are listed below. None of the proposed

techniques support the concept composibility factor [3, 4] when comparing two concept trees.

1.2.1 Existing Meaning Representation Technologies

There are several search engine frameworks exist today. They are based on the following meaning representation models. Some of them are shown below.

1.2.1.1 Boolean Model

Boolean search model are designed using Boolean algebra. It uses *exact matching* to match documents to user queries. The inability to identify partial matches leads to poor performance. Variations of Boolean search models like "Fuzzy Boolean engines" are derived which are based on fuzzy logic but this model still suffers from the problem that it cannot capture complex meanings. This model also cannot address two common problems of information retrieval process *synonymy* and *polysemy.* [2, 3, 6, 7].

1.2.1.2 Vector Space Model

This framework uses the *vector space model* developed by Gerard Salton [6, 10]. This model transforms text documents into numeric vectors and matrices then employ matrix analysis techniques to classify, retrieve, rank documents. The documents and query are represented by vectors as below

$$\boldsymbol{d_j = (v_{1,j}, v_{2,j}, \ldots, v_{t,j})} \qquad \boldsymbol{q = (v_{1,q}, v_{2,q}, \ldots, v_{t,q})} \tag{1.1}$$

To compute similarity, cosine of the angle between the query vector and the document vector is computed. A cosine value of zero indicates no match and a value of one indicates exact match. Different weight assigning schemes (e.g. td-idf) [6,7] are used

for assigning weights to the individual basis vectors of the document/query vectors. This model can not address the semantic sensitivity of documents because it used a bag of words approach. [6]

Advanced vector space models like *Latent Semantic Indexing* (LSI) [11] address the problems of *synonymy* and *polysemy* and also can access the semantic structure in a document collection.

This method still cannot address the problem of capturing complex ideas of documents and unsuitable for meaning based search framework.

1.2.1.3 Probabilistic Model

Probabilistic models [7] rank documents by their *odds of relevance,* the ratio of the probability that the document is relevant to the probability that the document is not relevant to the query. This model operates recursively and requires that the underlying algorithm guess at initial parameters, then iteratively try to improve this initial guess to obtain a final rankings.

This model is very complex and has scalability issues and is not suitable for Meaning based search framework.

1.2.1.4 Graph Based Model

Graphs [13] can be used to represent concept relations in a document. Each concept can be represented using a node and the concept relations can be represented using links between nodes. This technique can successfully represent the meaning of documents. This technique suffers from the fact that building and using such structures for computations is very expensive and non realistic. [3,14]

Accurate meaning comparison techniques require accurate meaning representation techniques. Before exploring the possible ways of representing a meaning lets discuss the notion of meaning briefly.

1.2.2 Meaning

Meaning is what humans think in mind when they hear some description of an object, or when they try to describe an object [8, 9]. For example to describe an object say a “car”, it can be done using a set of attributes of the object car, say the wheels, the engine, transportation means etc; each such description represents a concept. Human beings convey and comprehend meaning using concepts which are the mental representation of meaning. The complex description of car is represented using concepts wheels, engine, transportation etc. So comparison of meaning actually means comparison of concepts of the descriptions.

To capture the meaning of descriptions using concept trees we can represent the complex idea of the description using simpler hierarchical constructs of elementary concepts which require no further decomposition [3, 4]. An example is shown in Figure 4.

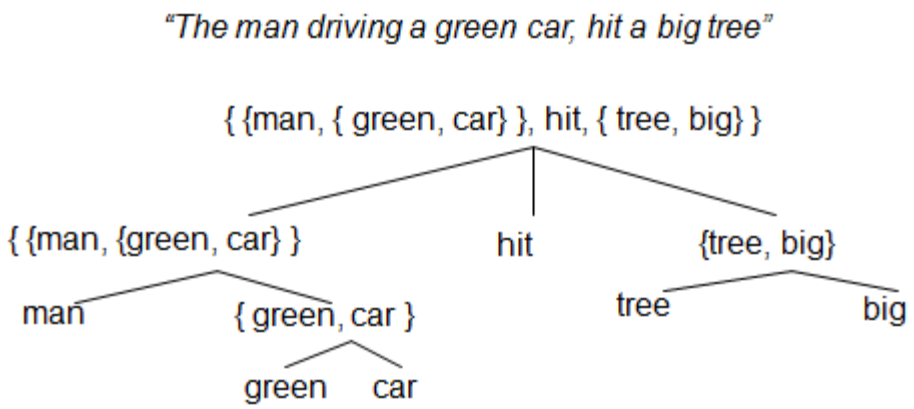

Figure 4 Concept tree to capture complex meaning

This concept tree structure can adequately represent the meaning of the description. For example the following two descriptions can be represented using the concept tree structures as shown in Figure 5.

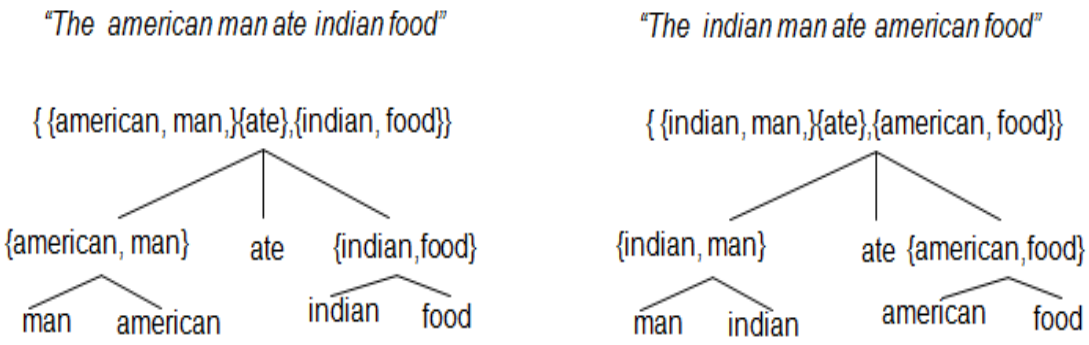

Figure 5 Concept tree distinguishing meanings

The next key challenge involved in Meaning based search is how to compare two concept trees. The comparison technique should consider the meaning compatibility factor of concepts when doing the comparison of two concept trees. Let's analyze if any existing tree comparison technique can be used for comparison.

1.2.3 Existing Tree Comparison Algorithms

These techniques are used to compute similarity between concept trees generated through classifiers, where concept refers to classes or categories in document collection or ontology. The classical tree similarity measuring approaches focus on the structural and geometrical characteristics of the trees. The degree of similarity between two trees is measured by the minimal cost of editing sequences that convert one tree into the other one from pure structural perspective. Some other techniques also take into account the knowledge information at concept tree nodes when doing a comparison.

1.2.3.1 Edit Cost (or Edit Distance)

Edit distance [14, 15] from one tree with reference to other tree is used to measure similarity of two trees. This technique mainly focuses on finding matches based on the pure structure or geometry perspective, without considering the conceptual semantics of the tree nodes in a knowledge context. This will not give accurate estimation of the similarity between two concept trees in our model.

1.2.3.2 Concept Taxonomy Modeling

Ontology has a tree structure that is modeling concept taxonomy [16]. A method was developed to measure the similarity between ontologies based on the notions of lexicon, reference functions, and semantic cotopy [16]. This method is based on an

assumption that the same terms are used in different ontologies for concepts but their relative positions may vary. This research did not take the structural characteristics of trees into consideration.

1.2.3.3 Tree Structure Mapping

It is another one of the often used methods. It proposes a mapping method that combines the similarity of the inner structure of concepts in different ontologies and the language similarity of concepts using lexical databases like WordNet [17]. This work did not handle cases of cross-layer mappings, which is common in tree mapping where similar terms may be placed in various layers within the trees and definitely a key requirement in our proposed model.

1.2.3.4 Tree Transformation

It is another technique that extends the classical tree editing operation based similarity measuring method to make it more applicable to compare trees that are representing concept structures [18]. Again this method is not suitable because meaning composibility factor [3, 4] is not taken into account.

Summarizing, to the best of our knowledge, no existing concept tree similarity computation technique exists that take into consideration tree structure of concepts and the composition of the concepts. Existing meaning representation models also cannot be used for the similarity computation.

1.3 Our Approach

So a new Tensor Model is proposed based on a new algebra theory that converts the *concept tree* into a *Tensor representation* [3, 4, 19]. This is the core process of the Meaning based search- retrieval framework. The *Tensor representation is a sum of scalar weighted polyads of basic basis vectors* which are elements at leaf nodes of the *concept tree*. A two level concept tree containing two child leaf concepts can be represented as below in Hilbert space as shown in Figure 6.

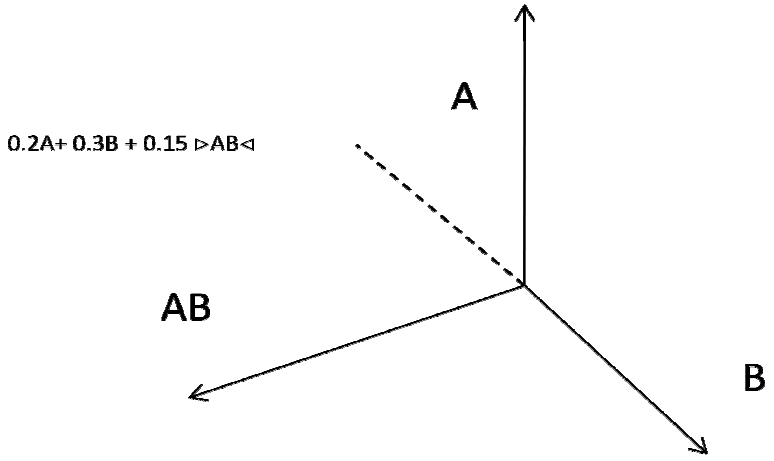

Figure 6 Tensor representation in Hilbert space for a concept tree

The tensor representation can be used for similarity computation by using the cosine similarity method analogous to the vector based model which is fast and efficient [3, 4]. Two normalized tensor representations can be used for similarity computation by taking the inner dot product of the tensors.

1.4 The Problem

This thesis explores the design of an algorithm to convert the Concept Tree into a Tensor representation using the proposed Tensor algebra theory. This thesis concentrates on designing and developing a tool in java using the proposed algorithm to generate Tensor representations from concept trees. This tool can be used for experiments to evaluate the Tensor model against vector space models. Our preliminary study showed that this approach outperforms the TF-IDF in 95% of the cases and Vector model in 92% of the cases [3, 4].

The thesis also describes the salient properties of the proposed Tensor Model based composition framework. The salient properties of the Tensor Model are represented as below.

Property I: Composition Information is included (Conjunction)

Property II: An incomplete set of elementary meanings can identify the composite meaning

Property III: Higher level compositions are more important

These properties are verified through simulations using an in house developed tool.

1.5 Notations

A Concept Tree (CT) is acyclic and directed n ary tree. For any two nodes u,v If $(u, v) \in E$ *(set of edges of CT)*, we call u a parent of v and v a child of u, denoted as $u = \text{parent}(v)$ or $v = \text{child}(u)$. The set of all children of node u is denoted as $C(u)$. The

intermediate nodes of the tree represent intermediate tensors of the sub-tree rooted at that node.

The following conditions are satisfied by any concept tree:

1. The root node does not have parent node.

2. Any node in CT other than the root has one and only one parent node.

3. There is a unique directed path composed by a sequence of elements in *E* from the root to each of the other elements in CT.

4. Each intermediate node has an associated co-occurrence set H, which defines the composition rules of the child nodes.

2. CONCEPT TREE TO TENSOR CONVERSION PROBLEM

In this section, we introduce the problem of finding a suitable algorithm for converting a concept tree structure to a Tensor representation. The goal of this conversion algorithm is to convert the concept tree into a tensor that can retain the compositions of the concepts in the concept tree and at the same time can be used effectively in similarity computations. This algorithm is developed from the Tensor algebra theory which addresses the above design rationales. In this section, we will explain details of Tensor algebra theory; follow it up with the algorithm and its implementation.

2.1 Tensor Algebra

The tensor algebra theory is designed to address the conjunction, disjunction compositions of the concepts of the concept tree. The tensor algebra theory [3, 4] expresses a *concept tree* as a *Tensor* in Hilbert space. The Tensor is represented by set of basis vectors, which comprises of the basic basis vectors (elementary concepts at leaf nodes) and polyadic combinations of the basic basis vectors (composite concepts at non-leaf nodes) [3,4]. These polyadic combinations represent the conjunction of basic basis vectors. The final Tensor representation of the concept tree is a sum of scalar weighted polyadic combination of basic basis vectors which are elementary concepts in the domain ontology.

The semantic similarity between two *concept trees* is given by the cosine product of their tensors representations. This technique is similar to what is used to find document similarity in *Vector space models*. A higher cosine product value indicates higher similarity between *concept trees* and thus the objects represented by the concept trees are semantically more close to each other.

To retain the compositions of concepts during Tensor generation two binders and a co-occurrence set is defined to carry out the transformation of concept tree to Tensor representation. A brief description of the binder algebra and co-occurrence set is given as below.[3,4].

2.1.1 Definition of $[\cdot,\cdot,\ldots]$ Binder

For case of one, two and three arguments we define:

$$[\,A\,] \equiv A$$

$$[A, B] \equiv AB + BA$$

$$[A, B, C] \equiv ABC + ACB + BAC + CAB + BCA + CBA$$

AB denotes a dyadic tensor product, ABC denotes a triadic tensor and a polyadic tensor [2,5] is denoted by juxtaposition (e.g., ABCD...). In general, AB ≠ BA. This definition can be expanded for a general case of "n" arguments, where the sum of product form has all permutations of arguments: A, B, C, etc.

2.1.2 Definition of $\{\cdot,\cdot,\ldots\}$ Binder

For one, two and three arguments:

$$\{A\} \equiv \frac{\sqrt{h_A} * [A]}{\sqrt{h_A^2}} = A$$

$$\{A, B\} \equiv \frac{\sqrt{h_{AB}/2} * [A, B] + \sqrt{h_A} * [A] + \sqrt{h_B} * [B]}{\left\| \sqrt{h_{AB}/2} * [A, B] + \sqrt{h_A} * [A] + \sqrt{h_B} * [B] \right\|}$$

$$\{A,B,C\} \equiv \frac{(\sqrt{h_{ABC}/6} * [A,B,C] + \sqrt{h_{AB}/2} * [A,B] + \sqrt{h_{BC}/2} * [B,C] + \sqrt{h_{AC}/2} * [A,C] + \sqrt{h_A} * [A] + \sqrt{h_B} * [B] + \sqrt{h_C} * [C])}{\left\| (\sqrt{h_{ABC}/6} * [A,B,C] + \sqrt{h_{AB}/2} * [A,B] + \sqrt{h_{BC}/2} * [B,C] + \sqrt{h_{AC}/2} * [A,C] + \sqrt{h_A} * [A] + \sqrt{h_B} * [B] + \sqrt{h_C} * [C]) \right\|}$$

This binder encompasses all possible combinations and permutations of arguments. The resultant tensor is also normalized and used as an elementary tensor to be incorporated for next higher level of composition.

2.1.3 Co-occurrence Set "H"

Each instance of $\{\cdot,\cdot,\ldots\}$ binder has a corresponding set of co-occurring coefficients "H", having real valued scalar elements. A tensor having three child concepts "A", "B", "C" will have seven coefficients (e.g. H = set { h_{ABC}, h_{AB}, h_{BC}, h_{AC}, h_A, h_B, h_C}), each of which indicates the importance of the corresponding polyad to represent the meaning of the composed concept.

For example, when only $h_{ABC} = 1$ and all other scalars $h_{AB} = h_{BC} \ldots = h_C = 0$, then the composed concept is the one which is given by a strict conjunction of A,B and C. Whereas the set $h_A = h_B = h_C = 1$ and $h_{ABC} = h_{AB} = h_{BC} = h_{AC} = 0$ represents disjunction composition. A mix of all these extremes is possible by suitable choice of co-occurring coefficients. Rules that guide assignment of these values can be codified and made accessible along with composition templates. These parameters are normalized by $(n!)^{1/2}$, where "n" is the number of arguments in $\{\cdot,\cdot,\ldots\}$ binder. [3,4]

Delimiter vectors "▷" and "◁" are introduced between tensors which are at different tree levels. The delimiter vectors point toward the tensor which belongs to a lower tree level. For example, instead of "CAB" and "ABC" we write "C▷AB" and "AB◁C". The use of delimiter vectors ensure that trees having same leaves but different composition do not have similarity beyond which is contributed by the individual leaves as shown in Figure 7. The ordering and combination of the leaf tensors and the delimiter vectors "▷" and "◁" in the polyadic products retains the information about the tree structure [3,4].

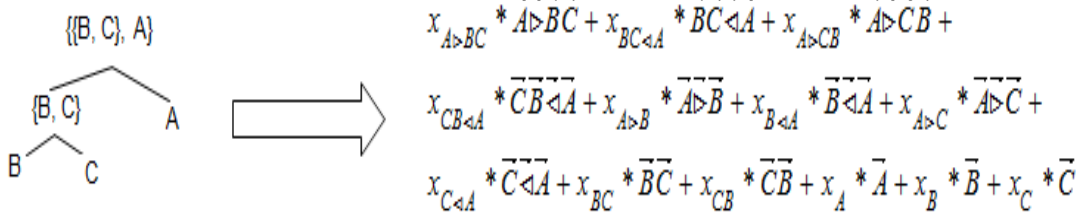

Figure 7 Tensor representations with delimiters "▷" and "◁"

2.2 Modified Tensor Algebra

A modified Tensor Algebra theory is developed to improve the performance of the Tensor model. The modified algebra will generate fewer basic vectors for final tensor representation.

2.2.1 Modified Binder Algebra & Co-occurrence set

Two modified algebraic binder connectives are proposed to improve the tensor generation process and generate fewer basis vectors in final tensor representation [3]. The new binders are **(1)** $[\cdot\cdot\cdot]$**;** and **(2)** $\{\cdot\cdot\cdot\}$, with following notations

1. Basic basis vectors with lower case alphabets with arrow on top e.g $\vec{a}$
2. Scalar co-efficient with lower case alphabets with no arrows s_i
3. Tensors as capital letters
4. hsh($\vec{a}$) is hash value of the string representation of basic basis vector $\vec{a}$

2.2.1.1 Definition of $[\cdot\cdot\cdot]$ Binder

For case of one, two and three arguments we define:

$$[\vec{a}] \equiv \vec{a}$$

$$[\vec{a},\vec{b}] \equiv\rhd \vec{a}\vec{b} \lhd \text{, if hsh}(\vec{a}) > \text{hsh}(\vec{b})$$
$$\equiv\rhd \vec{b}\vec{a} \lhd, \quad \text{if hsh}(\vec{b}) > \text{hsh}(\vec{a})$$

$$[\vec{a},\vec{b},\vec{c}] \equiv\rhd \vec{a}\vec{b}\vec{c} \lhd \text{ if, hsh}(\vec{a}) > \text{hsh}(\vec{b}) > \text{hsh}(\vec{c})$$
$$\equiv\rhd \vec{b}\vec{a}\vec{c} \lhd \text{ , if hsh}(\vec{b}) > \text{hsh}(\vec{a}) > \text{hsh}(\vec{c})$$
$$\cdot$$
$$\cdot$$

AB denotes a dyadic tensor product, ABC denotes a triadic tensor and a polyadic tensor [A,B,C,…] is denoted by juxtaposition (e.g., ABCD...). In general, AB ≠ BA. This definition can be expanded for a general case of "n" arguments, where the sum of product form has all permutations of arguments: A, B, C, etc.

$$[A,B,C,\ldots..] = \sum s_{a,i}\, s_{b,j} \ldots..[\vec{a}_i, \vec{b}_j, \vec{c}_{k,\ldots...}]$$

2.2.1.2 Definition of $\{\cdot,\cdot,\ldots\}$ Binder

For one, two and three arguments:

$$\{A\} \equiv \frac{\sqrt{h_A} * [A]}{\sqrt{h_A}} = A$$

$$\{A, B\} \equiv \frac{\sqrt{h_{AB}} * [A, B] + \sqrt{h_A} * [A] + \sqrt{h_B} * [B]}{\sqrt{h_{AB} + h_A + h_B}}$$

$$\{A,B,C\} \equiv \frac{(\sqrt{h_{ABC}} * [A,B,C] + \sqrt{h_{AB}} * [A,B] + \sqrt{h_{BC}} * [B,C] + \sqrt{h_{AC}} * [A,C] + \sqrt{h_A} * [A] + \sqrt{h_B} * [B] + \sqrt{h_C} * [C])}{\sqrt{h_{ABC} + h_{AB} + h_{BC} + h_{CA} + h_A + h_B + h_C}}$$

This binder encompasses all possible combinations and permutations of arguments. The resultant tensor is also normalized and used as an elementary tensor to be used for next higher level of composition as shown in Figure 8. The "h" values indicate the importance of a composition in the final tensor representation.

Polyadic combination of basis vectors are represented as concatenated strings, each of which represents individual basic basis vectors. hsh($\vec{a}$) represent the hash value of the string that represent the basis vector $\vec{a}$. For implementation 128 bit MD5 hashing is used. [3]

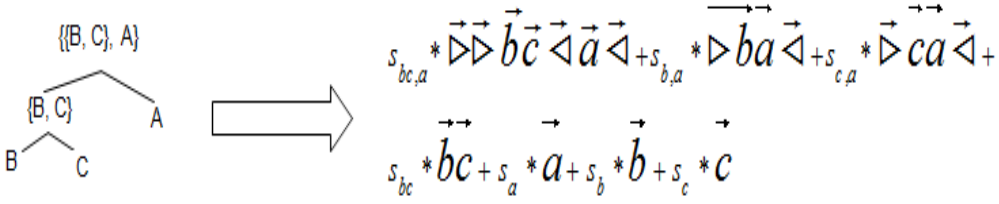

Figure 8 Tensor representations with delimiters "▷" and "◁" using new binder

The use of new binder generated fewer number of basis vectors as shown in Table 1 in the final tensor representation and requires fewer iterations. The table below shows a comparison of the number of basis vectors generated using the old binder and new binder. For our experiment analysis the Tensor representation algorithm uses the modified new binder functions for generating tensors.

Table 1 Superior performance of new binder

Sl. No	Tensor	# of leaves	Basic vectors (New Binder)	Basic vectors (Old Binder)
1	{{a,b},{c,d}}	4	15	40
2	{{a,b,c},{d,e,f},{g,h.i}}	9	511	21645
3	{ {a,b},{{c,d},{e,f}}}	6	63	364
4	{ {{a,b,c},{d,e,f},{g,h}}}	8	255	5598

2.3 Concept Tree to Tensor Generation Algorithm

To solve the problem of concept tree to Tensor transformation, we need to precisely define the concept tree structure and break it down to simpler problems. We can express a concept tree in terms of an N-ary tree where each node in the tree can have zero or more concepts. If a node is a leaf node, it will have one elementary concept representing the leaf node. If it's a non-leaf node, it will have links to multiple nodes, some of them could be elementary concepts or composite concepts (sub trees). This

concept tree structure can be defined using an abstract n ary tree. So from an algorithmic perspective the problem here is to transform the n-ary abstract tree using the binder functions in bottom up or top down manner and express the tree as a linear combination of the leaf node concepts and their compositions.

2.3.1 Approach

The tree expansion algorithm uses a bottom up approach to transform the n-ary abstract tree, i.e. for any level Ln, expand its child tensors at level Ln+1 using the binder before expanding the concepts at nodes in level n. Since each intermediate node represents an intermediate tensor for the concept sub-tree at that node, this recursive algorithm can be used to generate the Tensors at higher level nodes in the tree as shown in Figure 9.

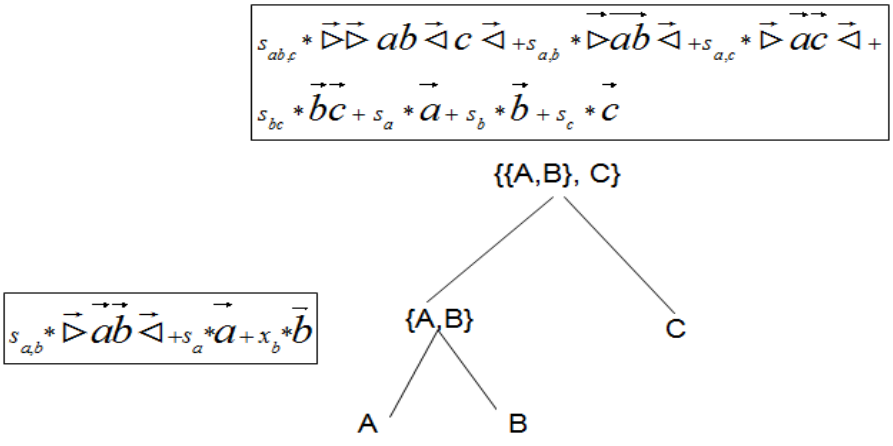

Figure 9 Tree to tensor expansion in bottom up fashion

In general, an N-ary *concept tree* structure is shown as above. Each node is a data structure that contains two kinds of information: the data for that node (tensor) and a collection of references to the next nodes in that sub-tree. If we closely observe the structure of the concept tree, each internal node in the tree represents an intermediate tensor of the sub concept-tree rooted at that node. Each leaf node can be thought of as a tensor having only one basis vector. This intuition gives an idea about the algorithm to compute the tensor representation using a recursive function which is ideal for a tree like data structure.

To define the steps involved in the recursion, and to hold the intermediate Tensors we define some useful data structures.\

2.3.2 Useful Data Structures

We defined some data structures for developing our tree expansion algorithm. They are listed below.

2.3.2.1 Product Container

$$\mathbf{P = X: A, B, C}$$

This container used to hold the individual basis vectors in the tensor representations corresponding to the [..] binder. This container has two components, one to hold the normalized scalar weight (X) associated with the basis vector and second one to hold the elementary concept tensors (A, B, C).

2.3.2.2 Sum Container

$$\mathbf{S = P_1, P_2, P_3 \ldots P_n}$$

This container will hold the composite tensor representations at intermediate nodes during expansion which will be used in the next higher level expansion. Each sum container corresponds to the {..} binder connective in Tensor conversion. The sum container at the root of the concept tree will store the final Tensor representation of the Concept tree.

2.3.2.3 Expansion List

$$\mathbf{L = P_1, P_2, P_3 \ldots P_n}$$

This list will hold the intermediate set of child tensors to be used for expansion in next higher level tensors. The details about how to use this data structure is given in detail with algorithm.

2.3.3 Algorithms

Let the *concept tree* be represented by a rooted n-ary tree CT. For any node w Let Child(w) and CT(w) represent the set of child nodes and the root of the concept tree rooted at w respectively. The co-occurrence set for any node w is given by Hset(w) which will store the composition factors for child tensors.

The pseudo code for the expansion algorithm is given as below.

2.3.3.1 Algorithm 1

ExpansionAlgorithm()

Algorithm: Tree Expansion Algorithm

Input: Concept Tree node w

Output: Tensor at node w

1. If node w is a leaf node

construct the leaf tensor

w.sumT.add(w)

return

2. Else prepExpansionList(w) of all tensors of child(w)

3. Normalize the final scalar weights of the tensor in the set.

4. Put the tensor set on w.sumT

5. End if

6. Return ct(w)

The algorithm for preparing expansion list can be extended from the idea of binders defined before [3,4]. To realize the expansion algorithm, we need to look into the detailed steps to see what is happening in each step of the algorithm. To generate a composed Tensor for a node at Level L, we need to list all possible compositions of Tensors at level L+1 of the concept tree. Now each Tensor is nothing but a set which contains a set of basis vectors, each having an associated scalar weight and a value that represents the basis vector concept of the Tensor. To find all possible compositions of n child tensor, we need to compute all possible combinations of all elements in n sets but not containing more than one elements of same set. To explain it in simple terms, we need to build an expansion list of size n, when there are n child tensors for any node. Each element in the list will belong to one of the child tensor sets and not two elements should be from the same set. The algorithm for building such a list can be explained

using the following examples. Here each term in the right hand side represent an expansion list. Examples:

{{AB}, {C}} = {[AB], C}} + {[A],{C}} + {[B],{C}}

{{AB}, {CD}} = {[AB], [CD]} + {[AB], [C]} + {[AB], [D]} + ….. + {[B], [D]}

To find the Tensor of the composition of tensors we need to prepare a list of possible compositions and expand the elements in the list using the binder defined in our algebra theory.

2.3.3.2 Algorithm2

prepExpansionList()

Algorithm Prepare Expansions List algorithm

Input: A concept tree node w, List L, currentChildCtr C

Output: Void

1. If C == w.ChildCount()-1

Call ExpandList with L

2. For count= 0 to child(w)[C].TensorCount() -1

3. L.add(child(w[C].Tensor(count))

4. C ← C+1

5. Call prepExpansionList with w,L,C+1

Algorithm to expand the list implements the functionalities of the binders of the tensor algebra. It uses the combination generation logic and generates all possible combinations of the elementLists of the expansion list. Every element of the expansion

list contains a set of basis vectors which needs to be expanded in tensor generation for next level tensor. The expandList algorithm operates on each of the elementLists of the expansion list and generates the final tensor. The tensor is normalized to set the proper scalar weights.

2.3.3.3 Algorithm3

Algorithm ExpandList

Input: List of compositions L

Output: w, concept tree node with Tensor basis vector terms

1.For every tensor composition in the List

2.Generate all possible combination basis vector Vt of elements in the List L using [binder { . .}]

a. List all possible combination of terms in L { . .} (Appendix A)

b. Get their MD5 hash values and sort using the value (system sort)

c. Append "<" and ">"terms

3. Compute the scalar co-efficient tensor terms Vt's in T by using Hsets

4. Add Vt to Tensor T

5. End - for

6. Return w

To compute the Tensor representation of the concept tree, the tree nodes are visited in post order traversal manner and generated the Tensor expression in a bottomup

fashion. The binder functions are implemented using tree visitor pattern so that future changes to binders can be incorporated easily.

2.4 Performance Evaluation

2.4.1 Space Requirement

Consider a node in the concept tree having "n" child concepts. The number of terms in the final Tensor for this node will have:

$$^{n}C_1 + {}^{n}C_2 + \ldots\ldots + {}^{n}C_n = 2^n - 1 \qquad (2.1)$$

Consider an intermediate tensor having two sets of Tensors containing elements n1 and n2. The composition according to the binder functions will generate elements in the final tensor equal to:

$$^{n1}C_1 + {}^{n2}C_1 + {}^{n1}C_1 * {}^{n2}C_1 = 2^{n1+n2} - 1 \qquad (2.2)$$

So for a complete N-ary concept tree, the final tensor will have $2^{n^d} - 1$, nodes, where d is the number of levels in the concept tree. The space requirement for the algorithm will be $\Theta\ (2^{n^d} - 1\ )$ for holding the final tensor and the intermediate stack for Depth First Traversal.

Space requirement to hold the expansion list is n. So the upper bound on the memory requirement is $\Theta\ (2^{n^d} - 1) + \Theta\ (\mathbf{n})$ for a "d" level complete n-ary tree.

2.4.2 Time Complexity

The timing requirements for the above algorithm will be dominated by the function that generates the binder {..}, i.e. generating the combinations. Here for generating the combinations the algorithm proposed by Kenneth H. Rosen is used [20].

Complexity of the algorithm is given by Θ ($2^{n^d} - 1$)$^{\mathbf{n}}$ **+ 2$^{\mathbf{n}}$** for a complete n-ary tree of depth d.

2.4.3 Scalability Analysis:

With controlled vocabulary, with leaves that represent composite meanings used, the concept tree size can be limited with number of leaves can be less than 15. These trees can be used for representing meaning properly and will generate Tensors with basis vectors (< 10^4). We know that an *n*-ary tree *T* of depth d >=0. The maximum number of leaf nodes in *T* is n^d[21]. A value of n^d <=15 indicates that expected values for both n and d will be in the range less than equal to 4.

2.5 Experimental Setup

The proposed Tree to tensor algorithm is implemented in Java. The reason for choosing java over other languages is the advantages it has over other languages in terms of speed of implementation and the portability. The garbage collection is also effective as the algorithm here is quite memory intensive.

The application consists of a backend that does the tree to tensor conversion, and the front end supports user interaction. The backend system expands the in memory concept tree in a bottom up fashion and generates the final tensor expression .The frontend reads a concept tree in a specified format and output the final tensor expression to user.

The backend system runs the core tree expansion algorithm on the input concept tree to generate the final tensor representation. The design of the backend system creates an n-ary generic tree to hold the concept tree structure and runs the expansion algorithm

on this generic tree. A tree visitor pattern us used to implement the expansion algorithm.

The tree visitor pattern [22] visits the nodes of the concept tree in a post order traversal manner and expands the concepts using the algorithms above. During the expansion a threshold value is used to select tensor terms for next level of expansion. If the scalar weights fall below the threshold value, they are not used for expansion in the next level. This approach reduces the memory requirements for storing tensors considerably. The application does not store the intermediate node Tensors once the next level tensor is computed by forming the expansion list to remove memory overheads. There is an option to write the intermediate Tensors to output files which can be used for debugging purpose. The use of tree designer pattern makes the easy integration of new approaches and enhancements of the algorithm to be implemented seamlessly without affecting the other programming pieces like input/output.

For simulations, the final Tensors are stored in output files and compared using another algorithm which computes the inner dot product of two tensors. The basis vector values are matched by comparing their converted MD5 values. We demonstrated that the proposed tensor based model can represent meaning more precisely compared to existing techniques. The success of meaning representation model is evaluated against TF-IDF model.

We took four publications from Pubmed [23, 24] on gene-diabetes interaction studies, which are the objects in consideration and denoted by O_i in Table 2. The object pairs are ranked based on three schemes:

1. Human interpretation (ideal case)

2. Semantic similarity values from Tensor Model and

3. Similarity values given by the TF-IDF vector based approach.

The concept trees are generated manually for the four objects. Initially the objects are ranked based on human interpretation and considered the ideal case for similarity comparisons. The tensors are generated from these concept trees and compared for similarity. For TF-IDF implementation we used the PMC collection [23,24] for generating the weighted term-document matrix and ranked our objects based on weights obtained from the TF –IDF model based cosine similarity computations. Table 2 gives the object similarity ranks and the Kendal tau [25] correlation of the models.

Table 2 Superior performance of tensor based approach for object similarity rankings

Object pairs	**Semantic similarity rankings and (similarity values)**			
	Human ranking	**Tensor approach**	**TF-IDF approach**	**Conceptual Vector approach**
P_1	Rank 1	Rank 1 (0.864)	Rank 1 (0.278)	Rank 4 (0.442)
P_2	Rank 2	Rank 2 (0.689)	Rank 2 (0.226)	Rank 1 (0.653)
P_3	Rank 3	Rank 3 (0.557)	Rank 4 (0.208)	Rank 5 (0.395)
P_4	Rank 4	Rank 5 (0.443)	Rank 6 (0.203)	Rank 3 (0.521)
P_5	Rank 5	Rank 4 (0.525)	Rank 3 (0.162)	Rank 6 (0.376)
P_6	Rank 6	Rank 6 (0.317)	Rank 5 (0.130)	Rank 2 (0.608)
Kendall's $\boldsymbol{\tau}$	1	0.867	-0.333	0.067
Difference			Tensor-TF_IDF corr. diff.. = 1.2	Tensor-C_V corr. diff.. = 0.8

The normalized cumulative frequency distributions of tensor-TF_IDF correlation difference ("tensor-TF_IDF corr. diff.") and tensor-conceptual_vector correlation difference ("tensor-C_V corr. diff.") are presented in Fig. 10. This clearly shows that in 95% of the cases the tensor-TF_IDF correlation difference is greater than zero. Similarly in 92% of the cases the tensor-conceptual correlation difference is greater than zero as shown in Figure 10. This indicates that tensor based rankings follow human ranking with greater fidelity than the TF-IDF and conceptual vector based ones. Hence we can deductively conclude that tensor based descriptor represents meaning more precisely than TF-IDF and conceptual vector based descriptors.[3,4]

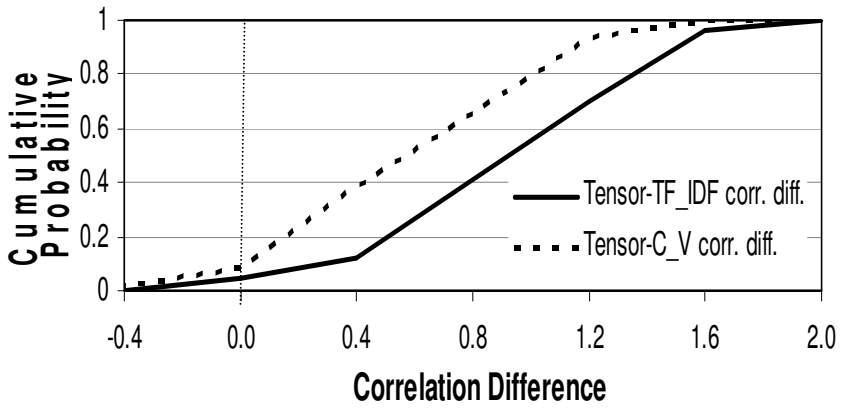

Figure 10 Cumulative freq. distribution of corr. diff.

3. VERIFICATION OF SALIENT PROPERTIES OF TENSOR MODEL

In this section, we describe the salient properties of a Tensor model framework. We built a simulation tool to generate synthesized concept trees and used these for verifying the tree properties and draw conclusions. In this section, first we will explain the properties of the Tensor model, follow it up with the design of the simulation tool with the algorithms used, and finally will give the simulation results that verifies the properties hold true.

3.1 Salient Properties of Meaning Based Framework

The Tensor based model has some useful properties:

Property I: Composition information is included (conjunction)

Property II: An incomplete set of elements can identify the composite meaning

Property III: Higher level compositions are more important

Details of these individual properties are given below:

3.1.1 Property I: Composition Information is Included (Conjunction)

This property indicates that tensor similarity measure can distinguish trees with similar leaves having different compositions, but vector based similarity cannot. These

properties infer that tensors should do a better job in discerning dissimilar compositions (trees) and meanings.

3.1.2 Property II: An Incomplete Set of Elements Can Identify the Composite Meaning

Two similar composite meanings may be expressed by two different but overlapping set of elementary meanings (i.e. they share many common elements) and yet they will be recognized as similar ones by the tensor model, as in case of vector model. This property is useful to identify similarity between contexts which are described by a slightly different set of elementary meanings.

3.1.3 Property III: Higher Level Compositions are More Important

The differences or similarities of elements at higher level compositions in a tree have larger impact on the similarity of the entire tree. All compositions are uniform mix of conjunction and disjunction compositions. The real world analogy of this property is that two objects will be considered similar if the big picture meanings of objects are similar even though the finer detailed meanings may be somewhat different.

To explain/verify these properties three metrics are used in the comparison of two concept trees.

3.2 Terminologies of Tree Comparison

3.2.1 Noise Ratio

The count of leaves not common between two concept trees CT1 and CT2 is called the “Noise”. The ratio is defined by the following formula

$$\text{Noise} / |\text{Leaves in CT1 U Leaves in CT2}| \qquad (3.1)$$

3.2.2 Overlap Ratio

The count of leaves present in similar locations in both concept trees is called "Overlap". The ratio is defined by the following formula

Overlap/ | Leaves in CT1 U Leaves in CT2| (3.2)

3.2.3 Displace Ratio

The number of leaves which are same in both trees but present in different locations is called "Displace". The ratio is given by

Displace/ | Leaves in CT1 U Leaves in CT2| (3.3)

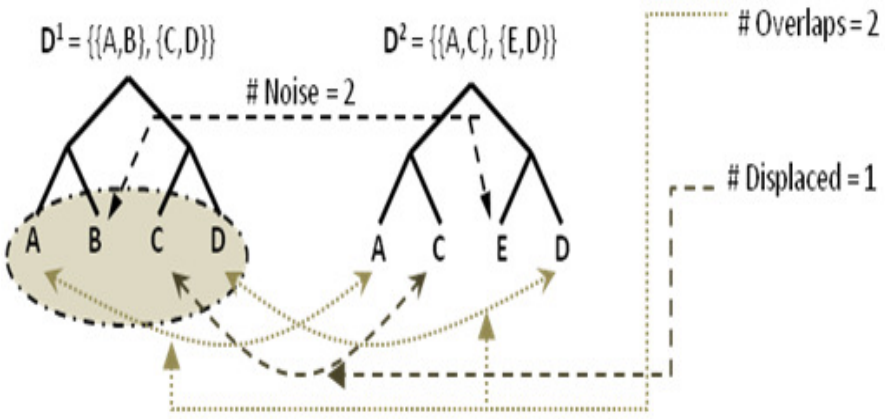

Figure 11 Noise, displacement, overlap in concept tree

The ratios defined here are analogous to the "Jaccard similarity" [26] measure used to compare two sets. Figure 11 shows the how the different ratios are computed.

3.3 Experimental Setup

To prove the properties of tensor model we designed a simulation tool to generate synthesized concept trees. We used these trees to generate tensors and used the

tensor representations to compute the similarity metric. We generated conceptual vectors for corresponding tensors by taking the basic basis vectors and then normalizing the scalar weights and by using random values for the vectors. Finally we did simulations and compared the vector similarity and tensor similarity with different noise/displacement/overlap factors. The design of the simulation tool and the results of simulations are explained below.

3.3.1 Simulation Tool to Generate Trees

A java based simulation tool is designed that can generate *concept trees* with a desired degree of randomness. This tree can be used as a reference tree and another tree can be generated from this reference tree by introducing noise/displacement/overlap. The algorithms for both the functionalities are explained below.

Algorithm GenerateRandomTree

Input: Tree Node "N" node , depth "d" , max possible degree range "r" for tree nodes, Co-occurrence set H

Output: Concept Tree "T"

1. *If depth d == 0 and T is not null return T*
2. *If N equals to root node create object T*
3. *Generate a random value C (child nodes) in the range of 2 to r (to avoid chaining in concept trees)*
4. *Call function to allocate C objects*
5. *For each child object C call GenerateRandomTree with d = d-1*

6. *Decrement d = d-1*
7. *Else (N is child node)*
8. *Check if N is a leaf node i.e if (d==0) assign node flag to Leaf L*
9. *Else generate Random children node C,*
10. *If C == 0, set depth =1, else if C==1 Merge Parent Child, call GenerateRandomTree with d = d*
11. *Else for each child object C call GenerateRandomTree with d = d-1*
12. *Decrement d = d-1*

Above algorithm will return a concept tree for a given depth range provided with nodes in the concept tree having a certain node degree. Before explaining the algorithm to generate concept tree with noise/displacement/overlap values we can look into the different kind of co-occurrence sets considered for this approach in our simulations. In the next section we are going to list the different kinds of co-occurrence sets considered for our experiments.

For generating concept trees, we need to provide co-occurrence sets for the concepts at different tree levels, which will describe the composition factors among the concepts in the concept tree. These co-occurrence sets will form the templates to generate concept trees in the simulator. The simulator is designed in such a way that the co-occurrence set to be used for experiments can be given as input to our Tree generation algorithm or the simulator can pick one of the available options randomly to generate concept trees. Both approaches is significant to run experiments which can be

tailored for particular scenarios. Here we are using six types of co-occurrence sets for our experiments.

3.3.2 Composition Templates

Six types of composition templates used for experiments

3.3.2.1 Pure Conjunction

This template set states that the conjunction of child concepts/tensors can only describe the meaning of object precisely. The individual concept tensors have no contribution to the final Tensor composition. For two concepts A and B, { h_{ab} =1, h_a =0, h_b =0 }.

3.3.2.2 Skewed Conjunction

This template set states that, the final composition of tensor is more skewed towards the conjunction of child concepts/ tensors. The individual elements of the set have some contribution to the final composed Tensor. For two concepts A and B, { h_{ab} =0.8, h_a =0.2, h_b =0.2 }.

3.3.2.3 Pure Disjunction

This template set states that the conjunction of child concepts/tensors cannot describe the meaning of object precisely. The individual concepts should be used for describing the object. For two concepts A and B, { h_{ab} =0, h_a =1, h_b =1 }.This template is very close to the way conceptual vector model generates vector of elementary leaves of a concept tree.

3.3.2.4 Skewed Disjunction

This template states that the conjunction of child concepts/tenors has a small significant contribution to the description of the final object. For two concepts A and B, $\{ h_{ab} = 0, h_a = 1, h_b = 1 \}$.

3.3.2.5 Uniform

This template is used to give equal weights to the compositions and to the individual child tensors. Thus the final tensor can have significant contributions from the individual concepts/tensors and from their compositions.

3.3.2.6 Random

This template gives random weights to the composition and to the individual concepts.

In our simulator we are generating trees using the above templates. When generating the tree with noise/displacement/overlap we use the first generated tree as reference, so both trees use the same template.

To implement the noise/displace/overlap tree generation algorithm, we have used a map container that will hold mappings of each leaf node and its parent node. This map is maintained for leaf nodes at each level. Because in concept tree we only deal with leaf nodes which will hold concepts this approach has a memory overhead to store mappings for all leaf nodes.

To generate a tree with noise, there can be three kinds of operations possible on the reference tree. Addition of a node, deletion of a node, or replacement of a node by another node. Addition and deletion operations introduce a single noise to the tree

whereas a replacement of a node by new node will introduce a double noise. So to generate a noise tree, one or more of the above mentioned operation is done on the input tree. The leaf node map is used to select a node at a tree level as victim node and operations are performed on its parent node. In some cases if a parent node contains only a single child node, the parent and child are merged to maintain the tree structure and avoid node chaining. The merge parent algorithm merges the child node with the parent node.

Similarly to compute a displaced tree, there can be two kinds of operations possible, swap of a node with another node belonging to two different sub trees or just a move of a node from one sub tree to another sub tree. Both operations can be done at any level of the tree. Again the leaf node map and merge parent child technique is used for generating the displaced tree.

3.3.3 Tree transformation Operations for Noise

3.3.3.1 Deletion Operation

Deletion operation of a leaf node will create a noise of one as shown below. The initial tree structure and the final tree structure after delete transformation shown in Figure 12.

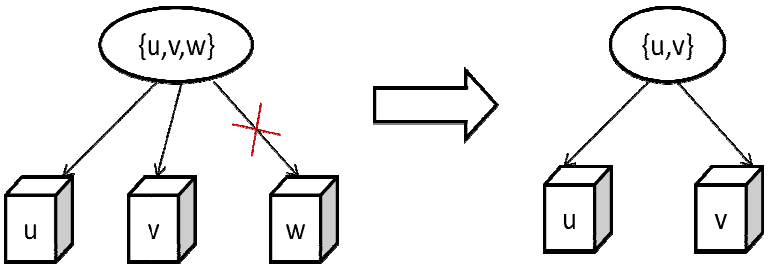

Figure 12 Deletion operation on a concept tree node for noise

3.3.3.2 Addition Operation

Adding a new leaf node to the tree will generate a noise value of one. The addition operation on a tree node is shown in Figure 13.

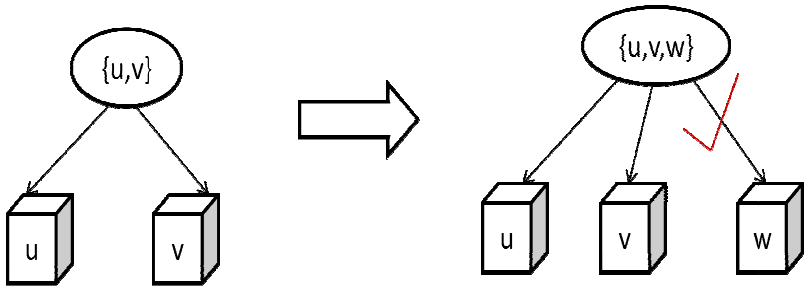

Figure 13 Addition operation on a concept tree node for noise

3.3.3.3 Replace Operation

A node replace operation will generate a noise of two. The node replace operation on a tree node is shown in Figure 14.

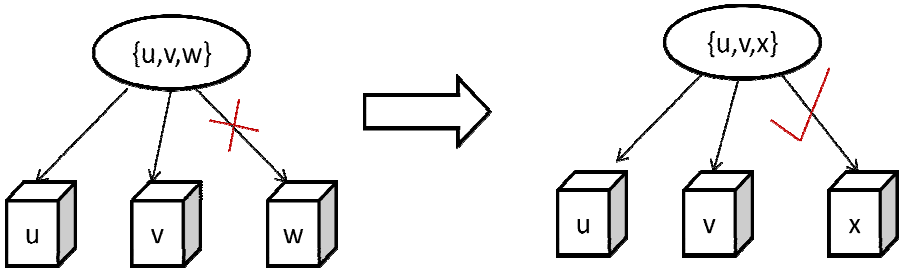

Figure 14 Replace operation on a concept tree node for noise

3.3.4 Tree Transformation Technique for Displacement

3.3.4.1 Addition Operation

A leaf node is added to a different parent node of the tree. The node add operation is shown below. It will generate a displacement value of one as shown in Figure 15.

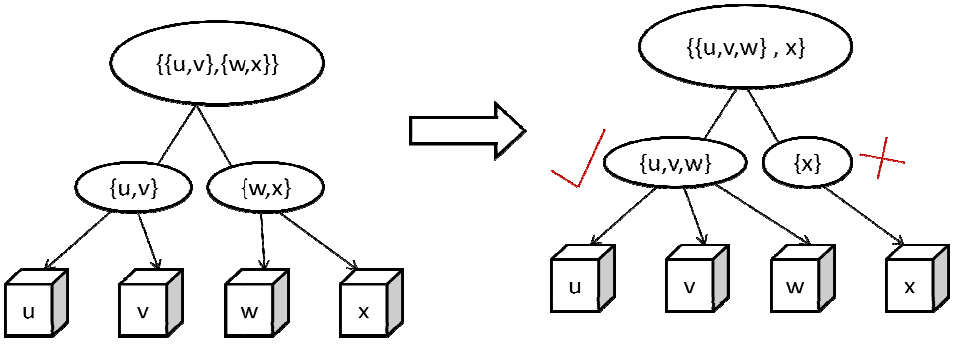

Figure 15 Addition operation on a concept tree node for displacement

3.3.4.2 Swap Operation

Two leaf nodes having different parent nodes swapped. This operation will generate displacement value equals to two as shown in Figure 16.

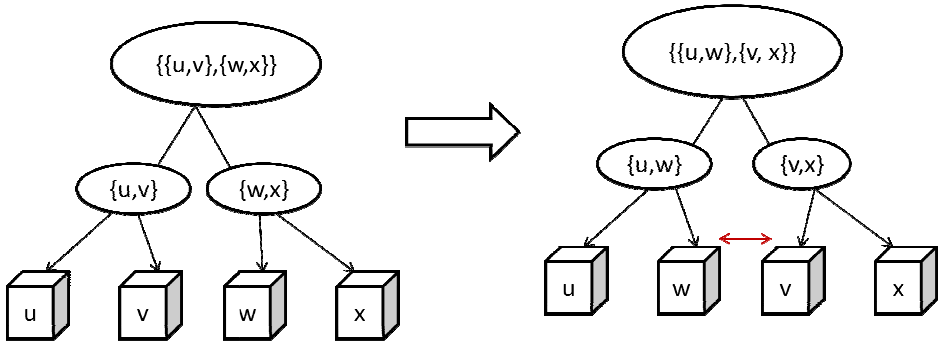

Figure 16 Swap operation on a concept tree node for displacement

Overlap will be computed by checking number of leaf nodes which did not get affected by noise or displacement.

Using the above techniques we can generate pairs of trees with Noise/Displacement/Overlap values and compare them to see how these factors affect the Tensor similarity computations.

3.4 Results & Evaluation

For our experiments we generated random trees with depth(d) in the range of 2 to 4 and node degree(n) in the range 2 to 4 in accordance with our assumption about the concept tree size [3,4] . We generated concept tree pairs (introducing displacement/noise) and used the Tree to Tensor application to generate Tensor

representations for concept trees and computed tree similarity dot product value. For the co-occurrence set option, each of the six available templates is used. The conclusions are drawn using hypothesis testing techniques. [27, 28]

Pure Conjunction

From the description of the templates provided earlier, the pure conjunction template is the one that deviates the most from the vector model. So during simulations, it is expected that for Pure conjunction template the similarity values will always be zero, because two descriptions with different structures will always give an absolute mismatch for this template. For operations of noise and displacement mentioned in section 3.4.4.1 and 3.4.4.2, the final tensors generated will not have any common terms, so the similarity value falls to zero. A sample size of 100 trees is chosen for the simulation. When there is a noise or displacement present and concept tree similarity is computed, it always gave a perfect mismatch as expected. This supports the property I hypothesis. For experiments we have taken sample size of 100 with the other five templates.

3.4.1 Property I

Property I claims that Tensor model captures the internal composition of the concepts in the trees. In other words, two descriptions though have the same concepts; if the tree structures (compositions) are different then Tensor model indentifies this by giving a similarity metric less than 1. But vector model cannot identify the compositions and will always give an absolute match of the two descriptions.

For proving the property I, all the six tree templates are used to generate concept tree pairs. The paired concept tree is generated using only displacement and no noise. All leaf concepts are given equal weights, and corresponding vectors of leaves are generated for the concept tree pairs. Similarity value is computed using the Tensor dot product of the concept tree tensor representations and vector similarity is computed by taking dot product of the normalized vectors. Single sample two tailed t-test is used for evaluation here. The hypnotized mean is chosen to be $\mu_0 = 1$ which indicates absolute similarity. The test proves tensor model similarities significantly deviate from this mean value. The proposed hypotheses are

$$\mathbf{H_0 : \mu_x = \mu_0.} \quad (3.4)$$

$$\mathbf{H_a : \mu_x \neq \mu_0.} \quad (3.5)$$

Table 3 T-test statistics for property I

Template	(Mean (M), SD, Sample Size(N))	t statistic	P-Value(two tail)
Skew Conjunction	(M=0.272, SD =0.0931, N= 100)	-78.23	$8x10^{-91}$
Skew disjunction	(M=0.8831, SD =0.0734, N= 100)	-15.94	$4x10^{-29}$
Pure Disjunction	(M=0.9557, SD =0.0665, N= 100)	-6.66	$2x10^{-9}$
Uniform	(M=0.6501, SD =0.0839, N= 100)	-41.71	$1x10^{-64}$
Random	(M=0.5307, SD =0.1206, N= 100)	-38.39	$9X10^{-62}$

The results from Table 3 show that the null hypothesis can be rejected ($t < -1.98$) at 0.05 significant level. So there is a significant difference between the mean of the sample and the hypnotized mean. The similarity value decreases with displacement. This is because, the structure change is been captured using compositions by Tensor model which successfully identifies two contexts with same elements but different meanings. Vector model gives an absolute match for all templates.

3.4.2 Property II

Property II states, similarity between contexts can be identified which are described by a slightly different set of elementary meanings. In other words small change in noise ratio has a small effect on the similarity between two contexts. To prove this property simulations are done using the above templates but keeping the displacement value to zero and introducing only noise. The similarity values by tensor model are measured against the noise ratio present in the concept tree structures. Noise ratio is an indicator of differences in the set of elementary concepts in the two concept tree. To verify the effect of noise ratio on similarity values of contexts, regression analysis is done to see the how the tensor similarity depends on the noise ratios. The following table shows the result of the analysis. The noise ratio is chosen as the independent variable and the tensor similarity as dependent variable. Noise ratio is in the range of (0, 0.8) used.

The results of regression for all the templates are shown in Figures 17-21.

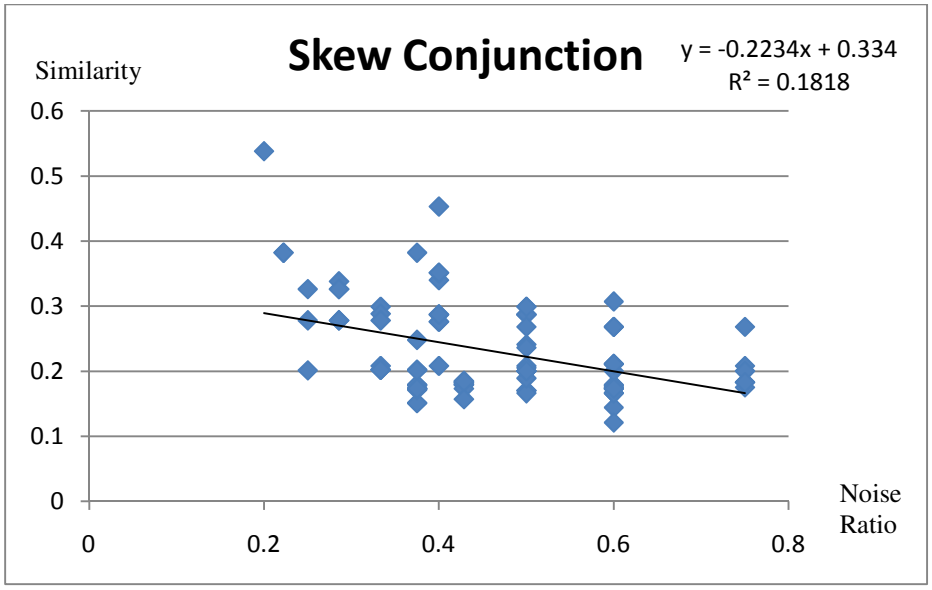

Figure 17 Regression Line for similarity and noise ratio for skew conjunction

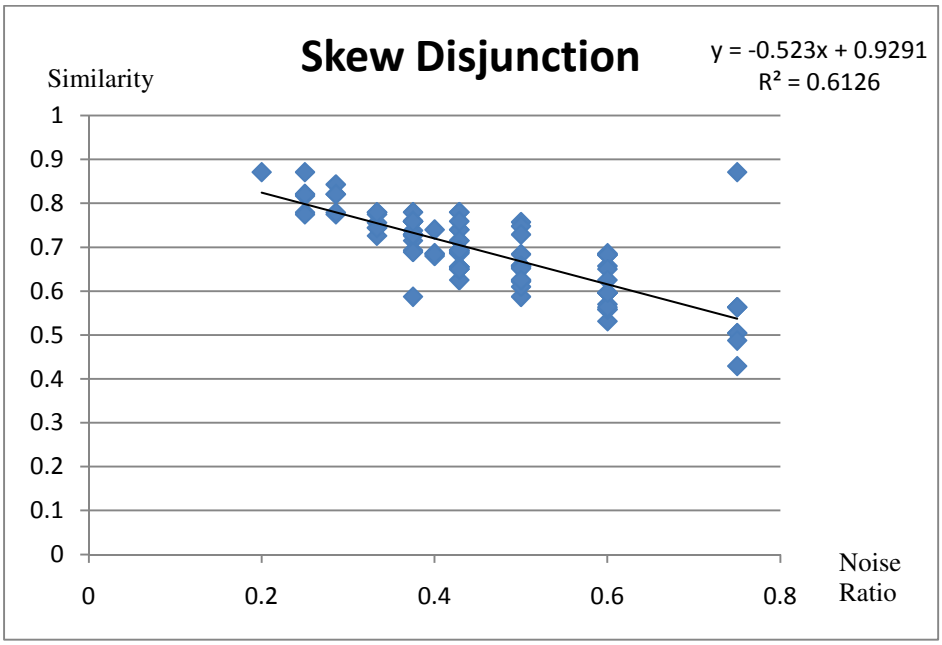

Figure 18 Regression Line for similarity and noise ratio for skew disjunction

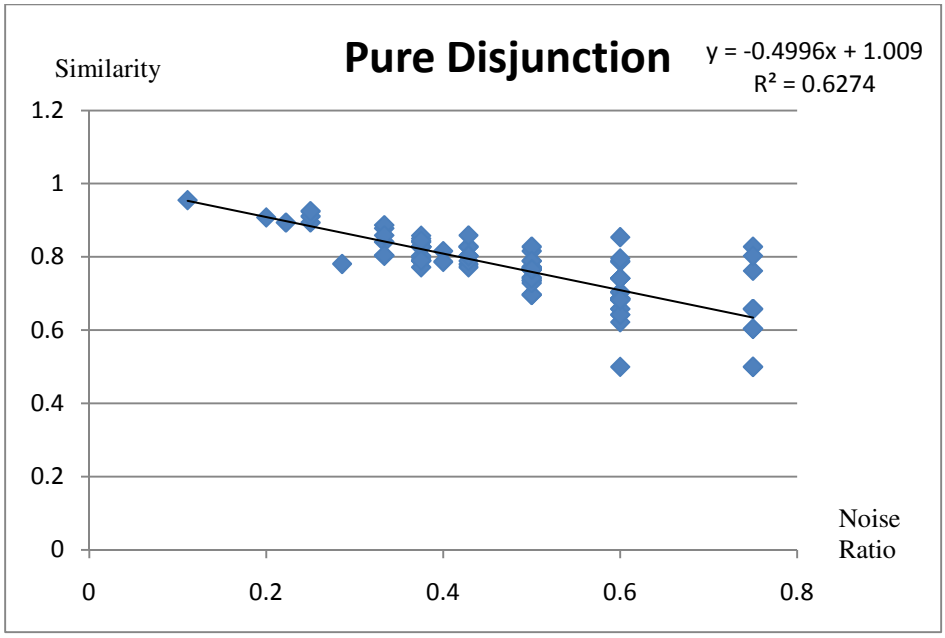

Figure 19 Regression Line for similarity and noise ratio for pure conjunction

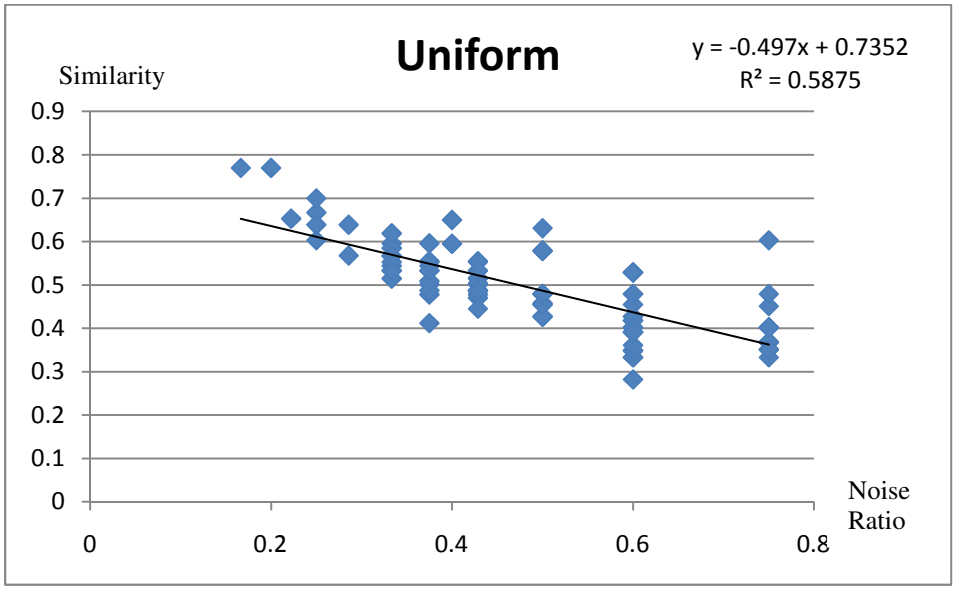

Figure 20 Regression Line for similarity against noise ratio with uniform

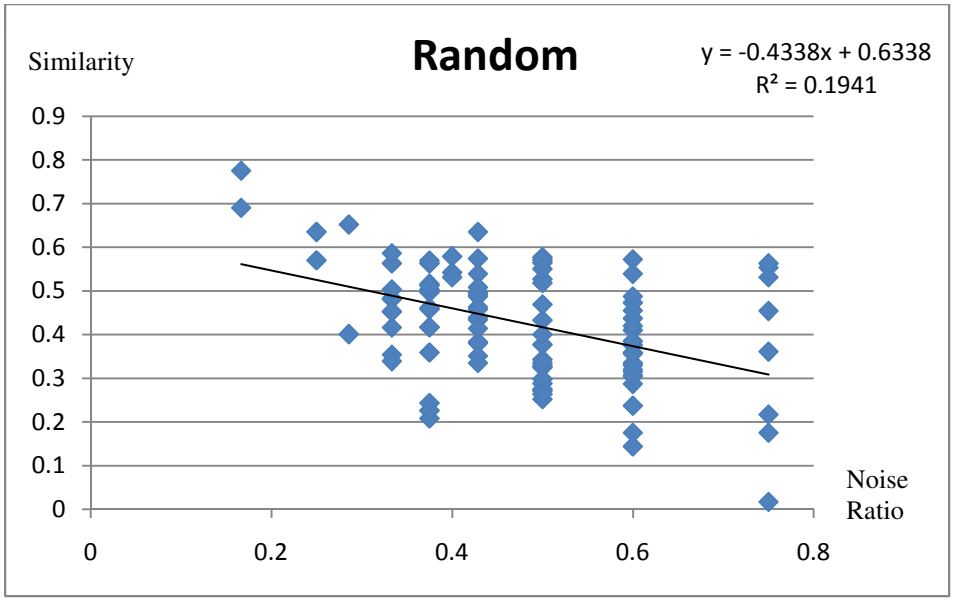

Figure 21 Regression Line for similarity against noise ratio with random

The results of the simulation showed that, with increase in noise ratio, similarity value decreases slowly. The line of regression for all the six templates has a very small slope with supports the property II of tensor model that small changes to noise ratio has very little effect on the similarity value. So tensor model can identify contexts with similar meaning but having slightly different elements in the concept tree.

3.4.3 Property III

Property III identifies the relative importance of higher level and lower level compositions in Tensor model. For verifying the property III, simulations are carried out with making displacement value to zero and noise is introduced at lower level nodes and higher level nodes. The similarity values are computed and evaluated for effect of noise at different levels. A noise at lower level means, overlap at higher level leaves and vice versa. The simulations are carried out for all six types of templates. Pure conjunction

template as expected gives an absolute similarity value of zero if noise is present so the other models are used for evaluation purpose. The results are shown below. A paired t-test with one sided tail is chosen for this simulation.

A paired t-test was performed to determine if higher level compositions are important.

3.4.3.1 Skew Conjunction

The mean similarity difference (M=0.08712, SD =0.0753, N= 100) was significantly greater than zero, t (99) =11.563, one-tail $p = 2.21 \times 10^{-20}$, providing evidence that the noise at lower tree level gives more similarity ($t > 1.66$).

3.4.3.2 Pure Disjunction

The mean similarity difference (M=0.09289, SD =0.001562, N= 100) was significantly greater than zero, t (99) =59.479, one-tail $p = 2.74 \times 10^{-79}$, providing evidence that the noise at lower tree level gives more similarity ($t > 1.66$).

3.4.3.3 Skew Disjunction

The mean similarity difference (M=0.12263, SD =0.02424, N= 100) was significantly greater than zero, t (99) =50.590, one-tail $p = 7.51 \times 10^{-73}$, providing evidence that the noise at lower tree level gives more similarity ($t > 1.66$).

3.4.3.4 Uniform

The mean similarity difference (M=0.12643, SD =0.008706, N= 100) was significantly greater than zero, t (99) =145.2134, one-tail $p = 1.8 \times 10^{-117}$, providing evidence that the noise at lower tree level gives more similarity ($t > 1.66$).

3.4.3.5 Random

The mean similarity difference (M=0.22503, SD =0.11831, N= 100) was significantly greater than zero, t (99) =19.0203, one-tail p = 3.93×10^{-35}, providing evidence that the noise at lower tree level gives more similarity (t > 1.66).

From the results obtained in section 3.4.3.1 – 3.4.3.5 we can reject the null hypothesis (t >= 1.66) at 0.05 significant level. So the difference between the means of the two groups is significant. In other words, the results indicate that the observations clearly show that there is strong evidence in favor of the alternate hypothesis that the difference in means of the similarity values of the two groups is significant. Thus introducing a noise at higher level has a greater impact, or conversely the higher level compositions are more important.

4. CONCLUSIONS

This thesis explores the design and implementation of an algorithm to convert a concept tree to a Tensor representation which is amenable for similarity computation in Meaning based search model framework. The research also explores the various properties of the Tensor based model and a simulation tool is developed for verifying those properties. A heuristic evaluation of the algorithm indicated that the application developed based on the algorithm can support the necessary requirements but can be improved further.

4.1 Future Work

4.1.1 Concept Tree from Text

To get more accurate estimate of the performance of the Tensor model approach we need an efficient algorithm to automate the process of generating concept trees from textual descriptions. The efficiency of this algorithm will have a stronger impact on the overall performance of the Tensor model in giving more accurate results.

4.1.2 Salient properties

The salient properties needs to be verified though other hypothesis testing methods [28,29] to draw more concrete conclusions and compare the effect of noise, displacement and overlap on the tensor model

REFERENCES

[1] A. Biswas, S. Mohan, J. Panigrahy, and R. Mahapatra., "Intelligent semantic technologies for distributed search networks," *Technical Report, Department of Computer Science, Texas A&M University*, US, July 2008.

[2] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. New York: Cambridge University Press, 2008.

[3] A. Biswas, S. Mohan, A. Tripathy, J. Panigrahy, and R. Mahapatra, "Semantic key for meaning based searching," *The 3rd IEEE International Conference on Semantic Computing* ,Berkeley, US, 14-16 Sept 2009.

[4] A. Biswas, et al. "Representation and comparison of complex concepts for semantic routed network,*" in Proc. the 10th International Conference on Distributed Computing and Networking,* Hyderabad, India*,* pp.127-138, 3-6 Jan 2009.

[5] Wikipedia, "Search Engine Index," http://en.wikipedia.org/wiki/Index_%28search_ engine%29, accessed 01/01/2010.

[6] G. Salton, and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Information Processing & Management*, vol 24, no. 5, pp. 513–523. 1988.

[7] A. Langville, "The linear algebra behind Search Engines," *Journal of Online Math ematics and its Applications,* December 2005, http://mathdl.maa.org/mathDL/4/?pa =content&sa=viewDocument&nodeId=636&pf=1,accessed 01/01/2010.

[8] G. Murphy, "Comprehending complex concepts," *Cognitive Science,* vol.12, no.4, pp. 529–562, 1988.

[9] R. Rajapske., and M. Denham, "Text retrieval with more realistic concept matching and reinforcement learning," *Information Processing and Management*, vol.42, pp. 1260-1275, 2006.

[10] J. Mitchell, and M. Lapata, "Vector-based models of semantic composition," in *Proc. of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, Columbus, Ohio, USA, pp. 236-244, June 2008.

[11] Wikipedia, "Latent Semantic Analysis," http://en.wikipedia.org/wiki/Latent_Semantic_analysis, accessed 01/01/2010.

[12] G.L. Murphy, and D.L. Medin, "The role of theories in conceptual coherence," *Psychological Review*, vol. 92, no. 3, pp. 289-316, 1985.

[13] H Ogata, W. Fujjibuchi, S. Goto, and M Kaneshia "A heuristic graph comparison algorithm and its application to detect functionally related enzyme clusters," *Nucleic Acids Research*, vol. 28, no. 20, pp. 4021-4028, 2000

[14] K. Tai, "The tree-to-tree correction problem," *Journal of Association for Computing Machinary*, vol. 26, no. 3, pp. 422–433, July 1979.

[15] P. Bille, "Tree edit distance, alignment distance and inclusion," *Technical Report Series TR*-2003-23, ISSN 1660- 6100*, IT University of Copenhagen*, March 2003.

[16] A. Maedche, and S Staab, "Measuring similarity between ontologies," in *Proc. of 13th International Conference on Knowledge Engineering and Knowledge Management, Ontologies and the Semantic Web*, pp. 251-263, 1-4 Oct 2002.

[17] D. Yang, and D.M. Powers, "Measuring semantic similarity in the taxonomy of wordNet," in *Proc. 28th Australasian Conference on Computer Science*, Newcastle, Australia ,vol. 38, pp. 315- 322, 2005.

[18] Y. Xue, C. Wang, H.H. Ghenniwa, and W. Shen, "A tree similarity measuring method and its application to ontology comparison," *Journal of Universal Computer Science,* vol. 15, no. 9, pp. 1766-1781, 2009.

[19] F. Irgens, *Continuum Mechanics*. Bergen: Springer Berlin Heidelberg, Jan 2008, http://www.springerlink.com/content/g31626k3m0828404/fulltext.pdf,accessed 02/ 01/2010.

[20] K. H. Rosen, *Discrete Mathematics and It's Applications*, 4th ed. New Jersey: WCB/McGraw-Hill, 1999.

[21] B. R. Preiss, *Data Structures and Algorithms with Object-Oriented Design Patterns in Java.* Waterloo: Wiley, 1999.

[22] Wikipedia, "Visitor pattern," http://en.wikipedia.org/wiki/Visitor_pattern, accessed 01/01/2010.

[23] The Gene Ontology Consortium, "Gene ontology: tool for the unification of biology," *Nature Genetics*, vol.25, pp 25-29, May 2000.

[24] L. Knecht, "PubMed: truncation, automatic explosion, mapping, and MeSH headings," *NLM Technical Bulletin,* May-Jun, 1998, p. 302.

[25] H. Abdi, “Kendall rank correlation,” in N.J. Salkind, *Encyclopedia of Measurement and Statistics*. Thousand Oaks (CA): Sage. pp. 530-532, 2007.

[26] Wikipedia, “Jaccard index,” http://en.wikipedia.org/wiki/Jaccard_index, accessed 01/01/2010

[27] L. Green, “Hypothesis Testing,” http://www.ltcconline.net/greenl/Courses/201/hvptest/index.htm , accessed 03/01/2010.

[28] Statistics Online Tutorial, “StatTrek,” http://stattrek.com/, accessed 03/01/2010.

[29] M. Gilleland,”Combination generator,” http://www.merriampark.com/comb.htm, accessed 01/01/2010.

APPENDIX A

PROGRAM FOR GENERATING COMBINATIONS

This program is developed from the algorithm proposed by Kenneth H. Rosen. The implementation of the algorithm which will generate the next combination from a set of n elements taking r elements at a time is implemented as below. The implementation is in Java[30].

A certain k-combination from the set S = {1, 2, 3, ... , n} can be represented as a subset of numbers from S in increasing order. These k-combinations can be enumerated using lexicographic order. The next combination after {c1, c2, ... , ck} can be obtained as follows:

1. Find the last element ci in the given k-combination such that ci does not equal n-k+i. If no such element exists (anymore), you're done;
2. If such a ci exists as described in step 1, replace it with ci+1 and cj with ci+j-i+1, for j = i+1, i+2, ... , k.

For example, let S = {1, 2, 3, 4, 5} and the combination c = {1, 4, 5}. Now c1 = 1, c2 = 4 and c3 = 5. The last term ci such that it does not equal n-k+i, is c1 = 1. Increment it to obtain 2 and let c2 = c1+1 = 2+1 = 3 and c3 = c2+1 = 3+1 = 4 resulting in the next combination cnext = {2, 3, 4}.

```
public int[] getNext () {

    if (numLeft.equals (total)) {
      numLeft = numLeft.subtract (BigInteger.ONE);
      return a;
    }
    int i = r - 1;
    while (a[i] == n - r + i) {
      i--;
    }
    a[i] = a[i] + 1;
    for (int j = i + 1; j < r; j++) {
      a[j] = a[i] + j - i;
    }
    numLeft = numLeft.subtract (BigInteger.ONE);
    return a;
}
}
```

VITA

Name:	Jagannath Panigrahy
Mailing Address:	Dept. of Computer Science and Engineering, 3112-TAMU, Texas A&M University, College Station, Texas 77843-3112
Email address:	jagannath.panigrahy@gmail.com
Education	B.Tech., Computer Science and Engineering, National Institute of Technology, Trichy, India, 2006 M.S., Computer Science, Texas A&M University, 2010