Exploiting hierarchical domain structure to compute similarity — Source link

Prasanna Ganesan, Hector Garcia-Molina, Jennifer Widom

Institutions: Stanford University

Published on: 01 Jan 2003 - ACM Transactions on Information Systems (ACM)

Topics: Similarity heuristic, Semantic similarity, Intersection (set theory) and Cluster analysis

Exploiting Hierarchical Domain Structure to Compute Similarity[1]

Prasanna Ganesan, Hector Garcia-Molina, Jennifer Widom

Stanford University

{prasannag,hector,widom}@cs.stanford.edu

Abstract

The notion of similarity between objects finds use in many contexts, e.g., in search engines, collaborative filtering, and clustering. Objects being compared often are modeled as sets, with their similarity traditionally determined based on set intersection. Intersection-based measures do not accurately capture similarity in certain domains, such as when the data is sparse or when there are known relationships between items within sets. We propose new measures that exploit a hierarchical domain structure in order to produce more intuitive similarity scores. We also extend our similarity measures to provide appropriate results in the presence of multisets (also handled unsatisfactorily by traditional measures), e.g., to correctly compute the similarity between customers who buy several instances of the same product (say milk), or who buy several products in the same category (say dairy products). We also provide an experimental comparison of our measures against traditional similarity measures, and describe an informal user study that evaluated how well our measures match human intuition.

1 Introduction

The notion of similarity is used in many contexts to identify objects having common "characteristics." For instance, a search engine finds documents that are similar to a query or to other documents. A clustering algorithm groups together gene sequences that have similar features. A collaborative filtering system looks for people sharing common interests [GNOT92].

In many cases, the objects being compared are treated as sets or bags of elements drawn from a flat domain. Thus, a document is a bag of words, a customer is a bag of purchases, and so on. The similarity between two objects is often determined by their bag intersection: the more elements two customers purchase in common, the more similar they are considered. In other cases, the objects are treated as vectors in an n-dimensional space, where n is the cardinality of the element domain. The cosine of the angle between two objects is then used as a measure of their similarity [McG83]. We propose enhancing these object models by adding a hierarchy describing

[1]This material is based upon work supported by the National Science Foundation under Grant No. 0085896.

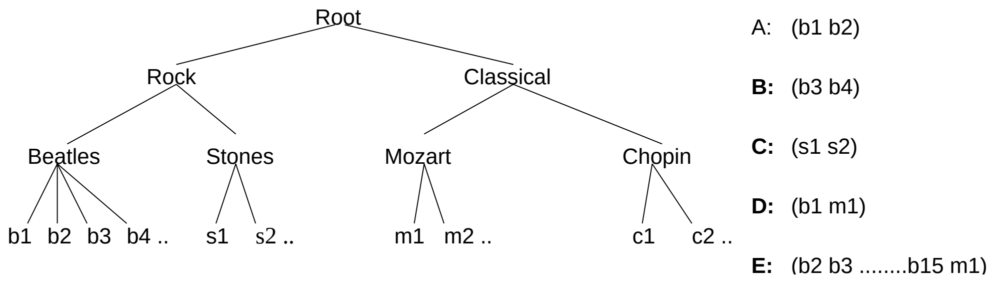


Figure 1: Music CD Hierarchy

the relationships among domain elements. The "semantic knowledge" in the hierarchy helps us identify objects sharing common characteristics, leading to improved measures of similarity.

To illustrate, let us look at a small 3-level hierarchy on the music CD domain, as shown in Figure 1. Let us say customer A buys Beatles CDs b_1 and b_2, B buys Beatles CDs b_3 and b_4, and C buys Stones CDs s_1 and s_2. If we were to use a similarity measure based on set intersections, we would find that the similarity between any two of A, B and C is zero. The Vector-Space Model would represent A, B and C as three mutually perpendicular vectors and, therefore, the cosine similarity between any two of them is again zero.

However, looking at the hierarchy of Figure 1, we see that A and B are rather similar since both of them like the Beatles, while A and C are less similar since both listen to rock music although they prefer different bands. The similarity between two CDs is reflected in how far apart they are in the hierarchy. In this paper, we develop measures that take this hierarchy into account, leading to similarity scores that are closer to human intuition than previous measures.

There are several interesting challenges that arise in using a hierarchy for similarity computations. In our CD example, for instance, customers may purchase CDs from different portions of the hierarchy: e.g., customer D in Figure 1 purchases both Beatles as well as Mozart CDs. In such a case it is not as obvious how similar D is to A or B or to other customers with mixed purchases. As we will see, there are multiple ways in which the hierarchy can be used for similarity computations, and in this paper we will contrast different approaches.

Another challenge is handling multiple occurrences (multisets) at different levels of the hierarchy. For example, say we had another user E who buys a lot of Beatles CDs as well as a Mozart CD m_1 (see Figure 1). The question is: Which of D or E is more similar to A? Customer D bought Beatles CD b_1, just like A. On the other hand, customer E did not buy that CD, but did buy a lot of other Beatles CDs. The traditional cosine-similarity measure favors multiple occurrences of an element. That is, if a query word occurs a hundred times in a document, the document is more similar to the query than one in which the query word appears only once. If we use this approach in our example, we would say that E is more similar to A than D is, because E buys 14 Beatles CDs, while D buys just one.

Unfortunately, it is not clear that this conclusion is the correct one. E is probably a serious Beatles fan, while A and D appear more balanced and similar to each other, so it would also be reasonable to conclude that D is more similar to A than E is. Thus, measures like cosine-similarity often do not provide the right semantics for inter-object similarity. This problem has, in fact, been observed earlier in the context of inter-document similarity [SGM95]. In this paper, we study various semantics for multiple occurrences, and provide measures that map to these semantics.

There has been a lot of prior work related to similarity in various domains and, naturally, we rely on some of it for our own work. In Sections 2 and 6 we discuss prior work in detail, but here we make some brief observations.

In our example we have seen that with traditional measures customers A, B and C have zero similarity to each other because their purchases do not intersect. When objects or collections are sparse, i.e., have few elements relative to the domain, intersections tend to be empty and traditional measures have difficulty identifying similar objects. There have been many attempts to overcome this sparsity problem through techniques such as dimension reduction [SKKR00], filtering agents [SKB+98], item-based filtering [SKKR01] and the use of personal agents [GSK+99]. We believe that using a richer data model (i.e., our hierarchy) addresses this problem in a simple and effective way.

Hierarchies, of course, are often used to encode knowledge, and have been used in a variety of ways for text classification, for mining association rules, for interactive Information Retrieval, and various other tasks where similarity plays a role [FD95, HF95, SM98, SA95]. Inspired by such prior uses of hierarchies, our goal here is to rigorously study how a domain hierarchy can be used to compute similarity, independent of a specific application, and to explore, compare and evaluate the various options that are available.

We believe that there are many domains in which hierarchies exist and can be exploited as we suggest here. Just to name a few examples, the Open Directory [Ope] is a hierarchy on a subset of pages on the web. Thus, we can compute the similarity of web users, for instance, based on a trace of the web pages they visit. In the music domain, songs can be organized into a hierarchy by genre, band, album, and so on. This hierarchy can then be used, say, to find Napster [Nap] users with

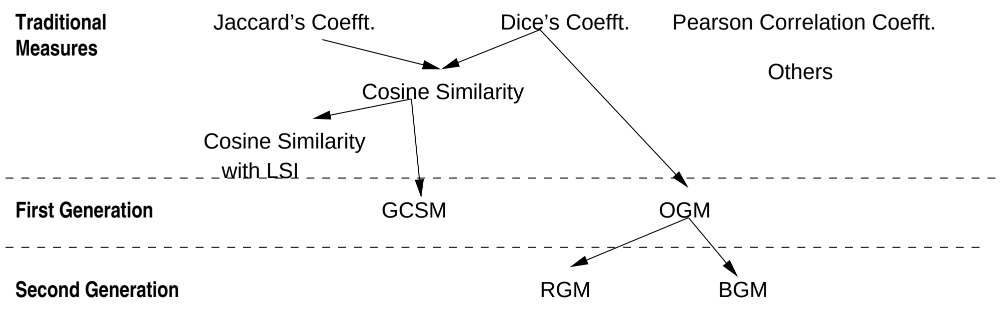


Figure 2: Evolution of Similarity Measures

similar tastes, and recommend new songs to them. In the document domain, we can use existing hierarchies such as WordNet [MRF+90] to compute document similarity. In all of these cases, our general-purpose extended similarity measures can be used to improve functionality.

In summary, the main contributions of this paper are the following:

- We introduce similarity measures that can exploit hierarchical domain structure, leading to similarity scores that are more intuitive than the ones generated by traditional similarity measures.
- We extend these measures to deal with multiple occurrences of elements (and of ancestors in the hierarchy), such as those exhibited in A and E in Figure 1, in a semantically meaningful fashion.
- We analyze the differences between our various measures, compare them empirically, and show that all of them are very different from measures that don't exploit the domain hierarchy.
- We report the findings of an informal user study to evaluate the quality of the various measures.

Figure 2 shows the evolution of the measures that we will discuss, and serves as a roadmap for the rest of the paper. Section 2 describes traditional approaches to computing similarity. Section 3 introduces our *First Generation* measures, which exploit a hierarchical domain structure and are obtained as natural generalizations of the traditional measures. Section 4 introduces the multiple-occurrence problem, and evolves the measures into our *Second Generation* measures. Section 5 is devoted to a comparison of these measures and their evaluation. Section 6 describes related work.

2 Traditional Similarity Measures

Given two objects, or *collections of elements* C_1 and C_2, our goal is to compute their similarity $sim(C_1, C_2)$, a real number in $[0, 1]$. The similarity should tend to 1 as C_1 and C_2 have more and more common "characteristics." There is no universal notion of which "characteristics" count, and hence the notion of similarity is necessarily subjective. Here we will define several notions of similarity, and discuss how intuitive they are.

2.1 The Set/Bag Model

In many applications, the simplest approach to modeling an object is to treat it as a set, or a bag, of elements, which we term a collection. The similarity between two collections is then computed on the basis of their set or bag intersection. There are many different measures in use, which differ primarily in the way they normalize this intersection value [van79]. We describe two of them here.

Let X and Y be two collections. *Jaccard's Coefficient*, which we denote $sim_{Jacc}(X,Y)$, is defined to be:

$$sim_{Jacc}(X,Y) = \frac{|X \cap Y|}{|X \cup Y|}$$

Thus, in Figure 1, $sim_{Jacc}(A,D) = \frac{1}{2+2-1} = \frac{1}{3}$. *Dice's Coefficient*, which we denote $sim_{Dice}(X,Y)$, is defined to be:

$$sim_{Dice}(X,Y) = \frac{2 * |X \cap Y|}{|X| + |Y|}$$

Once again referring to Figure 1, $sim_{Dice}(A,D) = \frac{2*1}{2+2} = \frac{1}{2}$. Other such measures include the *Inclusion Measure*, the *Overlap Coefficient* and the *Extended Jaccard Coefficient* [SGM00, van79].

2.2 The Vector-Space Model

The *Vector-Space Model* is a popular model in the Information Retrieval domain [McG83]. In this model, each element in the domain is taken to be a dimension in a vector space. A collection is represented by a vector, with components along exactly those dimensions corresponding to the elements in the collection. One advantage of this model is that we can now weight the components of the vectors, by using schemes such as *TF-IDF* [SB88]. The weight we assign to a dimension in a vector can be determined both by the number of occurrences of the element in that collection (*Term Frequency TF*), and by the relative importance of that element (*Inverse Document Frequency IDF*).

The *Cosine-Similarity Measure (CSM)* defines the similarity between two vectors to be the cosine of the angle between them, which is identical to the normalized inner product of the two vectors. This measure has proven to be very popular for query-document and document-document similarity in text Retrieval [SB88]. Again referring to Figure 1, and using uniform weights of 1:

$$sim_{Cos}(A,D) = \frac{\overrightarrow{A} \cdot \overrightarrow{D}}{|\overrightarrow{A}||\overrightarrow{D}|} = \frac{1 \times 1}{\sqrt{2}\sqrt{2}} = \frac{1}{2}$$

Collaborative-filtering systems such as GroupLens [RIS+94] use a similar vector model, with each dimension being a "vote" of the user for a particular item. However, they use the *Pearson Correlation Coefficient* as a similarity measure, which is given by the formula:

$$c(X,Y) = \frac{\sum_j (x_j - \overline{x})(y_j - \overline{y})}{\sqrt{\sum_j (x_j - \overline{x})^2 \sum_j (y_j - \overline{y})^2}}$$

where x_j is the value of vector X in dimension j, $\overline{x}$ is the average value of X along a dimension, and the summation is over all dimensions in which both X and Y are non-zero [RIS+94]. *Inverse User Frequency* may be used to weight the different components of the vectors. There have also been other enhancements such as default voting and case amplification [BHK98], which modify the values of the vectors along the various dimensions.

There are many other distance and similarity measures which have been defined for a variety of problems. Two of the most popular are *edit distance* and *Earth-mover's distance*. We will explain why they are inapplicable to our specific problem in Section 6.

3 The First Generation

We now describe two new measures we developed, based fairly directly on the traditional measures, that exploit a hierarchical domain structure in computing similarity. We first describe our model formally, define some associated concepts, and then proceed to develop the measures.

3.1 The Model

Let U be a rooted tree, with all nodes carrying a distinct label. We do not impose any restrictions on the shape of U: It can be arbitrarily unbalanced, and its leaves can be at different levels. Let L_U be the set of all labels in U. Let LL_U be the set of all labels on the leaves of U. LL_U is the element domain, on which there is a superimposed hierarchy described by U. In our music example, $LL_U = \{b_1, b_2, \ldots, s_1, s_2, \ldots, m_1, m_2, \ldots, c_1, c_2 \ldots\}$. A collection C is a bag whose elements are drawn from LL_U.

Let W be a function from LL_U to the set of real numbers. W is an *a priori* weight function on the leaves of U, which captures the relative importance of different elements. There are many ways

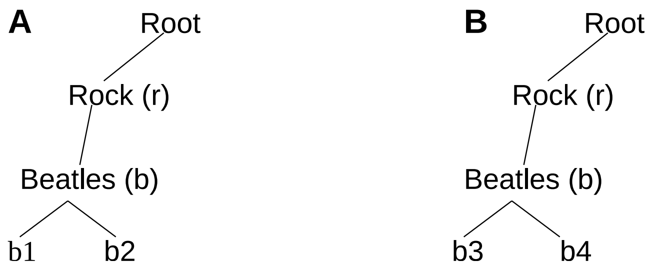


Figure 3: Induced Trees for Collections A and B

of deriving this weight function. It could be an *Inverse User Frequency* such as the one defined in [BHK98]. It could also be corpus-independent, and be determined by attributes of the elements, such as their cost (in monetary terms). Of course, the weight function also can be uniform.

Since there is a hierarchical structure imposed on LL_U, a collection C induces a tree, a subgraph of U that consists of the ancestral paths of each leaf in C. We refer to trees that are induced in this manner as *induced trees.* Notice that, since C is a bag, the induced tree might have more than one leaf with the same label. Figure 3 shows the induced trees for the collections A and B from Figure 1.

As is conventional, the *depth* of a node in the hierarchy is the number of edges on the path from the root of U to that node. Given any two leaves l_1 and l_2 in U, define the *Lowest Common Ancestor* $LCA(l_1, l_2)$ to be the node of greatest depth that is an ancestor of both l_1 and l_2. This LCA is always well-defined since the two leaves have at least one common ancestor—the root node—and no two common ancestors can have the same depth. In Figure 1, $LCA(b_1, b_2) = b$, while $LCA(b_1, s_1) = r$.

3.2 The Generalized Vector-Space Model

To illustrate how the Vector-Space Model can be generalized to take the hierarchy into account, consider Figure 1 again. Let us say that the unit vector corresponding to a leaf l is represented by $\overrightarrow{l}$. Now, according to the traditional cosine-similarity measure, all leaf unit vectors are perpendicular to each other, which means that the dot product of any two of them is zero. The dot product of a unit vector with itself is equal to 1.

We have already observed that b_1 is, intuitively, somewhat similar to b_3 since they are both Beatles CDs. Thus, if A buys b_1 and B buys b_3, we need to make this fact contribute something to the similarity of A and B, i.e., we want $\overrightarrow{b_1} \cdot \overrightarrow{b_3}$ to be non-zero. In the vector space, we want to assert that $\overrightarrow{b_1}$ and $\overrightarrow{b_3}$ are not really perpendicular to each other, since they are somewhat similar.

We use the hierarchy to decide exactly what value to assign to this dot product. For example, let us decide that $\overrightarrow{b_1} \cdot \overrightarrow{b_3} = \frac{2}{3}$, since they have a common ancestor that is two-thirds of the way down from the root. By a similar reasoning process, we let $\overrightarrow{b_1} \cdot \overrightarrow{s_1}$ be $\frac{1}{3}$. We let $\overrightarrow{b_1} \cdot \overrightarrow{m_1}$ continue to be 0 since they are in different sections of the hierarchy and don't really seem to have anything to do with each other, except for the fact that they are both music CDs.

Formally, let LL_U be the set $\{l_1, l_2, l_3, \ldots, l_n\}$. Let $Count_A(l_i)$ be the number of times l_i occurs in collection A. Then, collection A is represented by the vector $\overrightarrow{A} = \sum_{i=1}^{n} a_i \overrightarrow{l_i}$, where $a_i = W(l_i) * Count_A(l_i)$ for $i = 1..n$. This usage of weights is identical to the standard Vector-Space Model's. For any two elements l_1 and l_2, we define

$$\overrightarrow{l_1} \cdot \overrightarrow{l_2} = \frac{2 * depth(LCA_U(l_1, l_2))}{depth(l_1) + depth(l_2)}$$

This definition is consistent, since the right side of this equation always lies between 0 and 1. Note that the dot product is equal to 1 iff $l_1 = l_2$.

We continue to measure similarity by the cosine-similarity measure, except that we have now dropped the assumption that the different "components" of the vector are perpendicular to each other. If collection A is represented by the vector $\overrightarrow{A} = \sum_i a_i \overrightarrow{l_i}$ and B by the vector $\overrightarrow{B} = \sum_i b_i \overrightarrow{l_i}$, then:

$$\overrightarrow{A} . \overrightarrow{B} = \sum_{i=1}^{n} \sum_{j=1}^{n} a_i b_j \overrightarrow{l_i} . \overrightarrow{l_j}$$

Again, this equation is identical to the standard Vector-Space Model, except that $\overrightarrow{l_i} . \overrightarrow{l_j}$ is not equal to 0 whenever $i \neq j$. Finally, the cosine similarity between A and B is given by the traditional formula:

$$sim(A, B) = \frac{\overrightarrow{A} \cdot \overrightarrow{B}}{\sqrt{\overrightarrow{A} \cdot \overrightarrow{A}} \sqrt{\overrightarrow{B} \cdot \overrightarrow{B}}}$$

We call this measure the *Generalized Cosine-Similarity Measure (GCSM).*

3.3 The Optimistic Genealogy Measure

The Generalized Cosine-Similarity Measure from Section 3.2 is not the only, or even the most intuitive, way to exploit a hierarchy for similarity. Next we present a second, more natural and intuitive measure, and contrast it with GCSM. Intuitively, the *Optimistic Genealogy Measure*[2]

[2]The reason for the name becomes clear in the next section.

computes a "similarity contribution" for each element in one collection, and then takes the weighted average of these contributions to be the similarity between the two collections. The contribution of an element is determined by how good a "match" it has in the other collection.

Let C_1 and C_2 be the collections to be compared and let T_1 and T_2 be their induced trees as defined in Section 3.1. For any leaf l_1 in T_1, define $LCA_{T_1,T_2}(l_1)$ to be the ancestor of l_1 of greatest depth that is present in T_2, i.e., the lowest of the LCAs that l_1 shares with the leaves of T_2. This LCA provides an indication of how good the "best match" for l_1 can be. For example, for the trees in Figure 3, $LCA_{A,B}(b_1)$ is *Beatles*, since it is present in tree B, and is the lowest ancestor of b_1 that is present in B. (We abuse notation and let A and B refer both to the two collections and to their corresponding induced trees.)

Now define:

$$match_{T_1,T_2}(l_1) = \{l_2 \in C_2 | LCA(l_1, l_2) = LCA_{T_1,T_2}(l_1)\}$$

That is, $match_{T_1,T_2}(l_1)$ is the set of all leaves in T_2 that can be the "best match" for l_1. In Figure 3, $match_{A,B}(b_1)$ is the set $\{b_3, b_4\}$ since both elements match b_1 at its parent *Beatles*. Next, we define:

$$leafsim_{T_1,T_2}(l_1) = \frac{depth(LCA_{T_1,T_2}(l_1))}{depth(l_1)}$$

The value $leafsim_{T_1,T_2}(l_1)$ measures how similar l_1 is to its best match in T_2. If l_1 itself is present in T_2, then $LCA_{T_1,T_2}(l_1) = l_1$, and therefore $leafsim_{T_1,T_2}(l_1) = 1$. On the other hand, if no ancestor of l_1 except for the root is present in T_2, we have $depth(LCA_{T_1,T_2}(l_1)) = 0$ and, therefore, $leafsim_{T_1,T_2}(l_1) = 0$. In Figure 3, $leafsim_{A,B}(b_1)$ is $\frac{2}{3}$ and $leafsim_{A,B}(b_2)$ is also $\frac{2}{3}$.

Finally, for any two collections C_1 and C_2 with associated induced trees T_1 and T_2 respectively, we define the *Optimistic Genealogy Measure (OGM)* as:

$$sim(C_1, C_2) = \frac{\sum_{l_1 \in C_1} leafsim_{T_1,T_2}(l_1) * W(l_1)}{\sum_{l_1 \in C_1} W(l_1)} \tag{1}$$

This is just the weighted average of the individual *leafsim* values of the leaves in T_1. Note that since C_1 is a bag, the summation is over all members of the bag, and is not the set average. In our example, $sim(A, B)$ is also $\frac{2}{3}$, since the contributions from b_1 and b_2 are identical.

Note that OGM is, in general, asymmetric, i.e., $sim(A, B) \neq sim(B, A)$. If we desire to compute a symmetric similarity value between two collections C_1 and C_2, we could define it to be the average, the minimum, the maximum, or any other function of the two values, depending on what we desire.

sim	JC	DC	CSM	GCSM	OGM
A,B	0	0	0	0.8	0.67
A,C	0	0	0	0.4	0.33
A,D	0.33	0.5	0.5	0.65	0.67
B,C	0	0	0	0.4	0.33
B,D	0	0	0	0.52	0.5
C,D	0	0	0	0.26	0.25

Table 1: Comparison of the various measures

3.4 Discussion

Table 1 shows the similarity values computed by various traditional measures discussed in Section 2, as well as by GCSM and OGM, for the collections in Figure 1. JC stands for Jaccard's Coefficient and DC for Dice's Coefficient. The values shown are symmetric similarity values, with the average of the two asymmetric values being used for OGM. As motivated in Section 1, we would expect to find that customers A and B are more similar to each other than A and C. C and D should be even less similar. From Table 1, we see that both of our First Generation measures produce this result, while the traditional measures do not.

Intuitively, it is not clear whether $sim(A, D)$ should be higher than $sim(A, B)$. There is a case for saying that $sim(A, B)$ is higher, since both A and B are "pure" Beatles persons. One could also contend that A and D have a CD in common, while A and B have none, and, therefore, that $sim(A, D)$ ought to be higher. OGM gives them the same similarity values, while GCSM makes $sim(A, B)$ higher. The traditional measures claim that $sim(A, D)$ is higher, since they do not detect any similarity between A and B. GCSM and OGM can be tuned to adjust the conclusion in cases such as these. We discuss how to achieve this tuning in section 4.5.1.

3.4.1 Contrasting GCSM with OGM

Having seen how the First Generation measures fare on our simple example when compared with the traditional measures, we now examine the differences between GCSM and OGM in a little more detail.

- GCSM uses many-to-many matches, while OGM uses many-to-one matches. In GCSM, the similarity contribution of an element in one collection is gathered from all elements in the other collection that have a non-zero similarity to that element. On the other hand, OGM simply uses the best similarity score it can find for each element.

- GCSM is a symmetric measure, which means that we will not get high similarity scores if one collection is a subset of the other [SGM95]. OGM is an asymmetric measure, and conveys more information that may help us identify different semantic notions of similarity. For example, if we wanted to find an "expert" for a particular user A, i.e., someone who is knowledgeable about the things that A buys, we would look for a user B such that his purchases are close to a superset of A's purchases. Thus, $sim(A, B)$ would be very high, but $sim(B, A)$ might be fairly low.
- GCSM has worst-case complexity quadratic in the number of elements in the two collections. OGM has complexity linear in the number of nodes in the induced trees of the two collections.

4 Dealing with Multiple Occurrences – The Second Generation

The Vector-Space Model's approach to multiple occurrences of elements is a consequence of its origins in query-document similarity. The presumption is that, given a query word, a document that has 100 occurrences of the word is more relevant to the query than a document that has one occurrence of it. While this approach is reasonable for query-document similarity, it is not completely satisfactory for inter-document similarity, or, more generally, inter-collection similarity.

To see the problem, imagine three people X, Y and Z. Let's say that X buys one unit of some element e, Y buys 2 units of it, and Z buys 100 units of it (and all of them buy a few other, more-or-less-similar elements). Intuitively, X and Y are more similar than X and Z, since X and Y buy about the same number of units of e, while Z is quite different from the two of them. This conclusion is the exact opposite of that obtained by GCSM. OGM offers the same conclusion as GCSM since it, too, uses simple many-to-one matches. While one may not expect people to buy 100 copies of the same CD, there are many domains where such a situation does arise.

More importantly, the use of a hierarchy exacerbates the problem, since we no longer insist on exact matches. For example, let us look at Figure 1 and compute the similarity between A and E. According to OGM, $sim(A, E)$ is 0.75, while according to GCSM it is 0.89. Table 1 shows that $sim(A, D)$, according to the two measures, is 0.65 and 0.67 respectively. Thus, both measures claim that $sim(A, E)$ is higher than $sim(A, D)$. In this example, we don't have multiple copies of any one element, but we had a mismatch in the number of elements under the *Beatles* branch.

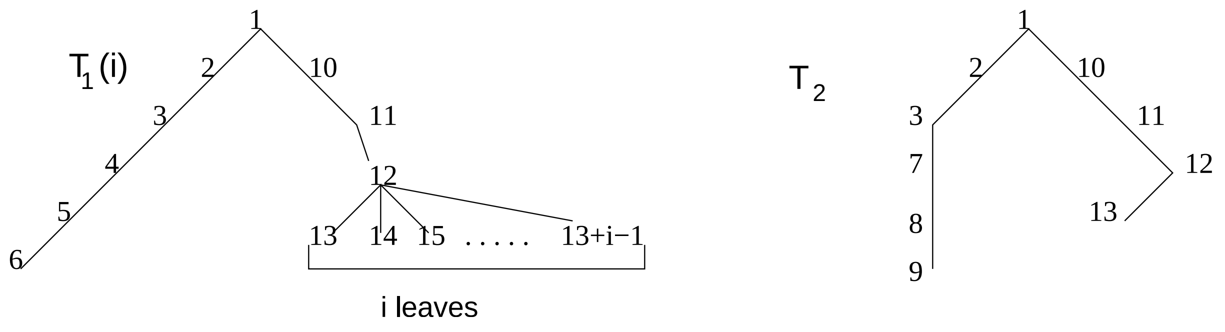


Figure 4: Illustrating the problems with multiple occurrences

i	OGM	GCSM	PGM	BGM(0.8)	RGM
1	0.7	0.7	0.7	0.7	0.7
2	0.717	0.717	0.47	0.667	0.638
3	0.725	0.703	0.35	0.620	0.617
4	0.730	0.690	0.28	0.573	0.606
∞	0.75	0.612	0.0	0.0	0.575

Table 2: Similarity between $T_1(i)$ and T_2

Thus, multiple occurrences at any level in the hierarchy can prove to be a problem.

In the rest of this section we will use an abstract example, shown in Figure 4, to explain the behaviour of the First Generation measures and the new measures that we propose. In this figure, we compare a family of collections represented by tree $T_1(i)$, for various i, to a collection represented by tree T_2. The weights of all leaves are taken to be 1. The right branch of $T_1(1)$ is identical to the right branch of T_2. As i increases, we add more and more leaves to the same branch of T_1 at node 12. We wish to see how $sim(T_1(i), T_2)$ changes as i increases.

Table 2 shows the (asymmetric) similarity values computed by the various measures as a function of i. For example, the first column shows the behaviour of OGM. We see that the similarity value progressively increases and converges to 0.75, which is what each additional leaf under node 12 contributes. According to our intuition, the similarity should decline as i increases, especially for large values of i.

The second column shows the behaviour of GCSM. We see that the similarity value goes up for a short while, and then eventually declines to 0.612. This pattern seems more promising but, actually, it too is poor. The crucial fact to note is that this value of 0.612 is still dictated solely by the contribution of each additional leaf, which is 0.75. But now, instead of the similarity simply

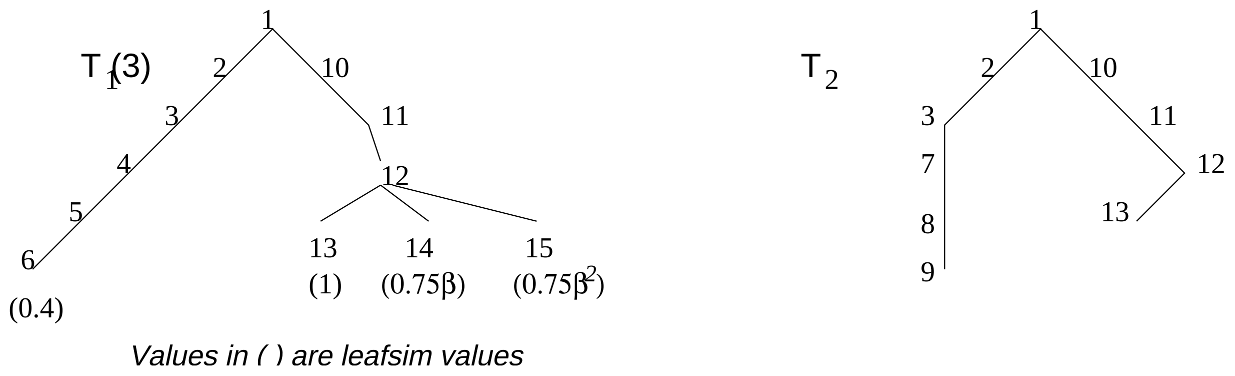


Figure 5: The Balanced Genealogy Measure

being equal to this value (0.75), it is actually proportional to the square root of this value.

Intuitively, the reason for this behaviour is that the magnitude of the collection corresponding to $T_1(i)$ also increases as i increases since $T_1(i)$ now has more "overlap" among its own elements. This increase has the effect of trying to lower the cosine similarity, but it is not strong enough to overcome the linear increase in the numerator of the formula for GCSM. We will provide a more detailed comparison of the semantic implications of the various measures at the end of this section. The rest of the columns in the table show the behaviour of the measures we will describe in the remainder of this section.

4.1 The Balanced Genealogy Measure

OGM admits of a simple generalization that solves the multiple occurrences problem. The general idea is to be less "optimistic" during similarity computation, and penalize many-to-one matches: if more than one leaf in the first tree gets its best match from one leaf in the second tree, we lower the similarity values that the duplicate matches contribute. Since we don't want to be too pessimistic in our similarity computation either, like the traditional measures are, we call this measure the *Balanced Genealogy Measure (BGM).*

BGM has a parameter β, a real number in $[0, 1]$, which controls the rate at which similarity decays with multiplicity of matches. To illustrate, consider the two trees $T_1(3)$ and T_2 in Figure 5. Each leaf of $T_1(3)$ is annotated with the *leafsim* value (recall Section 3.3) that BGM provides it. To see how these values are obtained, let us start with leaf 13 in $T_1(3)$. This leaf scores a value of 1 since 13 also exists in tree T_2. Next, we move on and try to find a match for leaf 14. The only possible match for 14 is, once again, leaf 13. In OGM, we would have given this match a score of 0.75. But now, we want to penalize leaf 14, since it matches with a leaf that has been matched once already. So, we give it a score 0.75β. For leaf 15, again, the best match available for it in T_2 is 13. Since 13 has been matched twice already, we give 15 a score $0.75\beta^2$. We then match leaf 6 with leaf 9 in T_2, giving it a score of 0.4. Finally, as usual we take the weighted average of these scores to arrive at an overall similarity score, which is 0.620 for $\beta = 0.8$.

The procedure we have outlined above is dependent on the order in which we examine the leaves of T_1. For example, if we had matched leaf 14 before leaf 13, leaf 14 would have received a score of 0.75 and 13 would have received a score 1 times β, thus lowering the overall similarity score. We define the similarity score produced by BGM to be the score generated by "optimal" matching, i.e., by the matching that maximizes the overall similarity score. We explain how to compute this score in the formal definition, next.

4.1.1 Formal Definition

Say we want to compute $sim(C_1, C_2)$, with C_1 and C_2 inducing trees T_1 and T_2 respectively. BGM proceeds as follows:

For each leaf l_1 in T_1, visited in *optimal order* (to be defined later):

1. Find a match l_2 in T_2. Recall that l_2 is a leaf in T_2 that provides the best LCA for l_1. If there is more than one possible match, pick that l_2 which has been matched the fewest times so far.
2. Increment l_2's match count. (Initially, all match counts are zero.)
3. Define:
$$optleafsim_{T_1,T_2}(l_1) = \frac{depth(LCA_{T_1,T_2}(l_1))}{depth(l_1)}$$
and:
$$leafsim_{T_1,T_2}(l_1) = optleafsim_{T_1,T_2}(l_1) \times \beta^{match\_count(l_2)-1}$$

The value $sim(C_1, C_2)$ is computed as the weighted average of the individual *leafsim* values, just as in OGM.

The *optimal order* is that order of visits of the leaves that leads to the highest possible similarity score computed according to this algorithm. If C_1 has n elements, the number of possible orderings of leaves is $n!$. So, we cannot afford to investigate every possible order and then pick the best one. Fortunately, it is possible to compute the similarity score according to the optimal order, with very little computational overhead. We first illustrate a simple case where all leaves are at the same depth and all leaf weights are equal.

The strategy we adopt is to look for matches in multiple phases. In the first phase, we look only for exact matches between leaves. In the second phase, we look for pairs of leaves with a common parent; in the third phase, pairs of leaves with a common grandparent, and so on. This strategy is guaranteed to produce the optimal score. To see why this strategy works, observe that we are looking for matches in the decreasing order of their *optleafsim* value contribution. If we think of BGM as being exactly identical to OGM, except that some of the *optleafsim* values are reduced, we see that the strategy always attempts to reduce the smallest possible *optleafsim* at each stage. For example, the match that produces the highest similarity score never gets reduced by β.

The strategy for the general case, where we have leaves at different depths and different leaf weights, is a generalization of the strategy outlined above. The key idea is to generate *leafsim* values in decreasing order of $optleafsim * W$, using a generalization of the multi-phase approach that we outlined above. This order is computed in a preprocessing step, and does not have to be generated for each individual similarity computation.

4.1.2 Computational Complexity

Let l_1 and i_1 be the number of leaves and internal nodes in T_1, l_2 and i_2 the number of leaves and internal nodes in T_2, h the height of the hierarchy, and b the maximum branching factor in T_2. Let $l = l_1 + l_2$ and $i = i_1 + i_2$.

We noted earlier that OGM has complexity $O(l + i)$. The computational complexity for BGM is higher than for OGM because, for BGM, we need to maintain state in the leaves of T_2 and use this state while computing similarity. There is also a slight overhead associated with computing the optimal order, because we need to examine $l_1 h$ nodes in T_1 instead of $l_1 + i_1$ nodes in the case of OGM.

In order to maintain state efficiently in T_2, we use priority queues to order the children of all the internal nodes. It can be shown that, due to the nature of updates to the various priority queues, the worst-case computational complexity is $O(lh(h + \log b))$.

In practice, $\log b$ would be much smaller than h which, itself, tends to be small in most domains. Also note that this bound is the worst case, and is realized only when computing the similarity between extremely dissimilar collections. In most applications, we would not be interested in the exact similarity value between such dissimilar collections, and will be able to prune the computation, thus achieving a much better computational complexity.

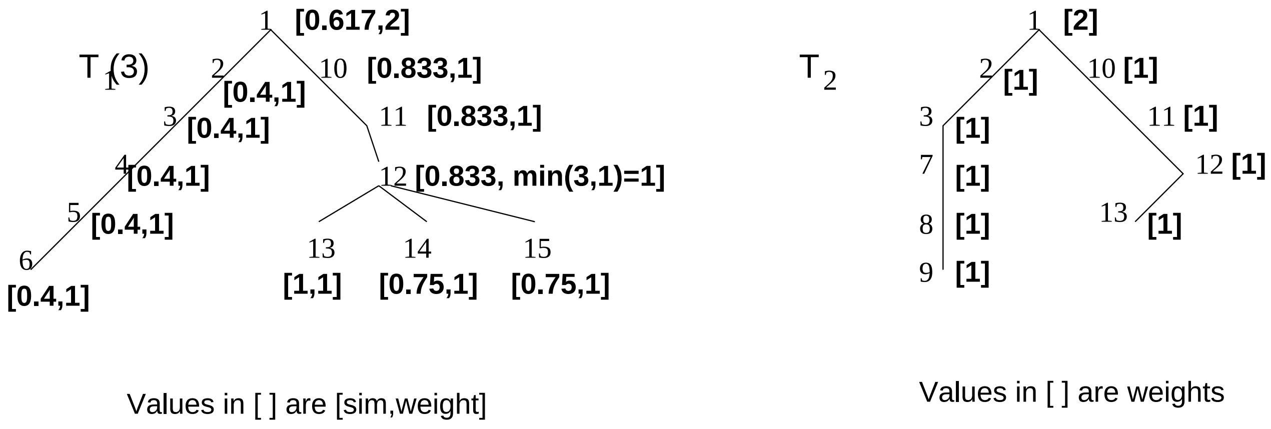


Figure 6: The Recursive Genealogy Measure

4.1.3 Discussion

First, notice that setting $\beta = 1$ instantiates BGM to OGM. At the other extreme, setting $\beta = 0$ is a pessimistic evaluation of similarity where we insist that no leaf in T_2 is matched more than once. We call this extreme the *Pessimistic Genealogy Measure (PGM)*. PGM and OGM provide the lower and upper bounds respectively on the similarity values computed by BGM.

We now look at the similarity scores computed by BGM for the example in Figure 4 for two different β values, 0 (PGM) and 0.8. Table 2 shows these values. We see that similarity declines to 0 in both cases, but it declines much faster with $\beta = 0$. This behaviour is no surprise, since β controls the degree of optimism of the measure. The important observation is that the similarity score actually does decline as i increases, which is what we set out to achieve.

4.2 The Recursive Genealogy Measure

Let us revisit the multiple occurrences problem. The problem, as we have seen, is that we tend to be too "optimistic" in our similarity estimates, which is unwarranted when we have many leaves in one tree matching just one leaf in the other tree. If we revisit the computation performed by OGM, shown in Equation 1, we see that there are two ways of solving this problem. The first is the approach adopted by BGM, namely lowering similarity for duplicate matches. Alternatively, we could leave the similarity values alone and, instead, lower the *weight* that we assign to these duplicate matches. This is the approach that we study now, called the *Recursive Genealogy Measure (RGM)*. In Section 4.3, we compare the semantic underpinnings of these two approaches.

We will, once again, use trees $T_1(3)$ and T_2, shown in Figure 6, to explain RGM. The similarity computation consists of two phases. In the first phase, we simply compute *leafsim* values just as in OGM. In Figure 6, they are the first element of the ordered pairs on the leaves of $T_1(3)$. To compute the overall similarity value for the two trees, we use a bottom-up computation on $T_1(3)$ to make the *leafsim* values flow to the top of the tree. The value that is obtained at the root node is the similarity between the two trees. The value at an internal node stands for the similarity between that subtree and the appropriate portion of T_2.

In order to perform this computation, we first need to define weights for the nodes in the two trees. We will take the weights of all the leaves to be 1 in this example. Let us first look at T_2. Here, the weight of an internal node is simply the sum of the weights of its children. Thus, the root has a weight of 2, while all others have a weight of 1 (shown in square brackets). The crux of the measure is in the assignment of weights to the nodes in $T_1(3)$. Weights are defined in $T_1(3)$ just as in T_2, with one exception. If a node in T_1 also happens to be in T_2, and its weight is lower in T_2, we use this lower value as its weight in T_1.

In the figure, the weights of the nodes are shown as the second element of the ordered pairs enclosed by [ ]. In the left subtree of $T_1(3)$, the weights are all simply 1. But in the right subtree, the weight of node 12 is not 3. We notice that 12 has a lower weight of just 1 in T_2. So, we assign it a weight 1 in $T_1(3)$, too. Notice that this assignment of weights in $T_1(3)$ captures the multiple occurrences. The fact that all leaves under 12 can match just a single leaf in T_2 is captured by assigning a weight of 1 to node 12.

Once these weights are assigned, we compute similarity by a simple bottom-up calculation. The similarity at any internal node is the weighted average of the similarity at all its children. Thus, in Figure 6, the similarity at node 12 is seen to be $\frac{1+0.75+0.75}{3} = 0.833$ (shown as the first element of the ordered pair). The similarity we see at the root is the actual similarity value between the two trees, which is 0.617 in this case.

4.2.1 Formal Definition

For any tree T and any node n in T, let $C_T(n)$ be the set of all children of n in T. Let $W_T(k)$ be the weight of node k in tree T. We will shortly explain how to compute $W_T(k)$, given our original weight function W which is defined only for the leaves of trees.

Let C_1 and C_2 be the two collections under comparison, and let T_1 and T_2 be their associated trees, as usual. We first define the weights to be associated with all the nodes in tree T_2. We then define the weights for all the nodes in T_1.

$$\begin{aligned} W_{T_2}(n) &= W(n) \text{ if } n \text{ is a leaf of } T_2 \\ &= \sum_{c \in C_{T_2}(n)} W_{T_2}(c) \text{ if } n \text{ is an internal node of } T_2 \\ &= \infty \text{ otherwise} \end{aligned}$$

We have defined the weights of nodes not in T_2 to be ∞ for notational convenience.

$$\begin{aligned} W_{T_1}(n) &= W(n) \text{ if n is a leaf of } T_1 \\ &= min(\sum_{c \in C_{T_2}(n)} W_{T_1}(c), W_{T_2}(n)) \text{ if n is an internal node of } T_1 \\ &= 0 \text{ otherwise} \end{aligned}$$

For any internal node in T_1, its weight is determined both by the sum of the weights of its children in T_1, say p, and by the weight of the same node in T_2, say q. Although we have chosen to use $min(p, q)$ as our weight, we could, in general, use any function, although functions that return a value between p and q make the most sense. We discuss the effect of this choice in Section 4.3.

Let $sim_{T_1,T_2}(n)$ denote the similarity value "at" a node n in tree T_1. The similarity between the two trees $sim(T_1, T_2)$ is given by:

$$sim(T_1, T_2) = sim_{T_1,T_2}(root(T_1))$$

For all nodes n in T_1, we define:

$$\begin{aligned} sim_{T_1,T_2}(n) &= optleafsim_{T_1,T_2}(n) \text{ if } n \text{ is a leaf (defined in Section 4.1.1)} \\ &= \frac{\sum_{c \in C_{T_1}(n)} W_{T_1}(c) * sim_{T_1,T_2}(c)}{\sum_{c \in C_{T_1}(n)} W_{T_1}(c)} \text{ if } n \text{ is an internal node} \end{aligned}$$

4.2.2 Comparison

First, notice that the computational complexity of RGM is linear in the total number of nodes in the induced trees of the collections under comparison. This complexity is the same as OGM and better than BGM.

The last column in Table 2 shows the similarity values as computed by RGM for the trees in Figure 4. We see that the similarity value declines as i increases, which is the effect that was desired. We also notice that the similarity value declines slowly and does not eventually converge to zero; instead, it converges to a value 0.575. At first sight, this behaviour seems to resemble GCSM, which also declines and converges to a non-zero value. But there is a big qualitative difference between the values that they converge to, which we explain in Section 4.3. RGM is very different from GCSM as will also be seen from our experimental results in Section 5.1.

4.3 Summary and Discussion

The fact that we have proposed more than one measure, each of which handles multiple occurrences in its own way, is a natural consequence of the different possible interpretations of the idea of similarity. Reconsider our original example in Figure 1, particularly the similarity between A and E. Recall that A has two Beatles CDs, while E has 14 Beatles CDs and one classical music CD.

One way to look at the similarity of A and E would be to observe that a high percentage of E's purchases are Beatles CDs. Therefore, we could treat E as a "Beatles person." Since A is also a "Beatles person", we give them a very high similarity score. This interpretation is the one offered by OGM. GCSM uses an interpretation that is almost identical, but with one important difference. It observes that each of A's purchases is very similar to almost every one of E's purchases. The high similarity score resulting from this observation is tempered by the fact that E's purchases are, themselves, very similar to each other.

The BGM interpretation is influenced by the difference in size between A and E: The fact that E has 14 Beatles CDs while A has just 2 makes them somewhat dissimilar according to BGM. The fourth, and final, interpretation differs markedly from the first three. None of the first three interpretations were influenced much by the fact that E bought a Mozart CD. All of them were swayed primarily by the fact that the majority of the CDs bought by E were Beatles CDs. The RGM interpretation localizes the effects of the Beatles CD purchases, and is influenced by the other purchases of A and E as well.

It is not clear that one of these interpretations is always the "correct" interpretation. Quite often, it depends on the nature of the domain, the nature of the collections, and the exact semantic need. For example, if we knew that we wanted similarity of queries to documents, and we don't care too much about overlap between query terms, we would settle for the first interpretation (OGM). The second interpretation (GCSM) might be useful for longer queries, where we might take into account the fact that two of the query words are describing related concepts. For example, if we had both the words "car" and "bicycle" in a query, which also consisted of many other words, we might want to take into account the relationship between these words.

Choosing between the third (BGM) and fourth (RGM) interpretations is dictated by the relative importance of the coverage and distribution of elements. For example, in Figure 1 we could choose to ignore the fact that E bought a Mozart CD, as BGM does, as long as we care only about the distribution of the elements. If coverage is important, we do want to factor in E's Mozart CD, and the RGM interpretation permits us to do so. By suitably choosing the correct function to use in computing weights in RGM, we can pick the desired balance between coverage and distribution.

4.4 Other Extensions

4.5 Other Extensions

There are other extensions to the model and the metrics that we have omitted for ease of description. We provide a brief overview of some of them here.

4.5.1 Edge Weights

We can introduce edge weights into our tree model, assigning them *a priori*. Edge weights helps capture the relative importance of a 'concept leap' from a parent to a child. For example, the 'distance' between *Aerosmith* and *Hard Rock* may be smaller than the 'distance' between *Hard Rock* and *Rock*. Modifying the metrics to handle these edge weights is a straightforward exercise: We use distances from the root instead of node depth, when computing leaf similarities.

4.5.2 DAGs

Several of these metrics can be extended to handle *DAGs* rather than just trees. We just redefine the concept of the *LCA* to be that ancestor that provides the highest leaf similarity value. Generalizing RGM to handle *DAGs* is rather more complicated, since it relies on a bottom-up computation which we will have to generalize to DAGs.

4.5.3 Handling Weight Skew

In all our algorithms, we might be matching a leaf l_1 with a high weight with a low-weight leaf, say l_2. When we compute the overall similarity in BGM by computing a weighted average, we have simply used the weight of l_1. But, there is a case for using a lower weight for l_1 in the averaging process, if the leaf it matched had a much lower weight. This problem is, in fact, identical in spirit to the size skew problem. The simplest way to handle it is to introduce a new factor $f(W_{T_1}(l_1), W_{T_2}(l_2))$, a value between 0 and 1 that will modify the *leafsim* value that we assign to l_1.

The Recursive Genealogy metric already handles the weight-skew problem in a correct fashion.

5 Evaluation

We now proceed to evaluate our measures empirically. There are at least three types of questions one may pose:

1. How different are the measures from each other and from the traditional measures in practice? If all measures give roughly the same rankings, we might as well use the traditional measures. But if there are differences, can we characterize when the differences occur?
2. How well does each of them match human intuition? Would a human agree with the similarity rankings produced by our measures?
3. What measure is best or most appropriate for a given application?

In this paper we focus on the first two types of questions, since the third type is clearly application-dependent and very hard to answer. However, we believe that if a particular measure matches human intuition (item 2 above), it is likely to perform well in a variety of applications.

In Section 5.1, we provide detailed comparisons of the various measures, analyze where and how much the measures differ and, in the process, show that using a hierarchy produces results very different from those produced by traditional measures. In order to show that a hierarchy yields more intuitive similarity results, we rely on a user study, as detailed in Section 5.2.

In our evaluation, we choose Jaccard's Coefficient as representative of the traditional measures, and refer to it as the *Naive* measure. All of the traditional measures are extremely similar when compared against our First and Second Generation measures, so Jaccard's Coefficient is a good representative.

5.1 Experimental comparison of the different measures

For the experiments reported in this section, we used transcripts of undergraduate CS majors at Stanford as our data set. Each transcript is a collection of (*course,grade*) elements. The objective is to compute how similar two students are, on the basis of the courses they have taken and the grades they have obtained in those courses. There were a total of 403 transcripts, with an average of about 41 (*course,grade*) pairs per transcript.

The hierarchy consists of 6 levels: department, course level, course subject, course number, and a two-level grade classification, in that order. Changing this order leads to different hierarchies and, consequently, different semantics for similarity. For example, placing the grade levels at the top of the hierarchy would mean that we want to pay more attention to the grades that students get, rather than the courses they take, in determining the similarity between two students. Thus, the choice of the hierarchy reflects the semantic need of the application.

5.1.1 The Distance Measure

Given a similarity measure M and any collection X, we can generate a ranked list of collections $L_M(X)$, in decreasing order of similarity to X. In most cases, it is this ranked list that is important, rather than the actual similarity values that we compute. Moreover, most applications only care about the top portion of this list, say the Top K, whether it is in order to find the nearest neighbors of a given collection, or whether it is to return the Top K matches to the collection. We therefore define a *distance measure*[3] to compare similarity results on the basis of these ranked lists of collections.

Let S be the set of all collections. Let $rank_{M,X}(A)$ be the rank of collection A in list $L_M(X)$. Further, let $L_M(X)[i]$ refer to the collection that has rank i. Our distance measure compares the ranked lists generated by two different measures by imagining one of the measures as generating an "ideal" ranking. We then measure how much each collection is displaced from its ideal ranking by the second measure.

To illustrate, let us look at Figure 7. There are two measures, 1 and 2, that produce two different ranked lists of collections A, B, C, D and E, given some other collection X to compare against. We now want to quantify the difference between these two ranked lists. First, we notice that A has rank 1 in the first list, while it has rank 2 in the second. Thus, A contributes a displacement of 1 to the total distance between the lists. Next, we see that B has rank 2 in the first list, while it has rank 1 in the second. This is an upward displacement and we do not count it, because it is already captured by the fact that A, B, C and D are all pushed down a step by B's moving up. Similarly, C contributes a displacement of 1, while D and E contribute nothing. Thus, the total distance between the ranked lists is 2, and the average displacement is $\frac{2}{5} = 0.4$.

In the example above, we computed the distance over the whole list. Computing it over the Top K is done in the same manner, except that we only consider the top K collections of the first measure.

Formally, we define the *Top-K Distance* between measures M_1 and M_2, when used to compute

[3]not to be confused with our *similarity measures*

Metric 1	**Metric 2**
A	*B*
B	*A*
C	*D*
D	*C*
E	*E*

Figure 7: The Distance Measure

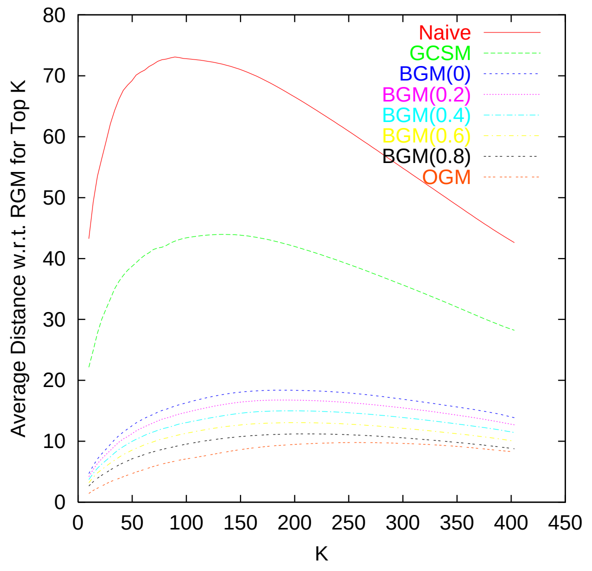


Figure 8: Average Top K distance w.r.t. RGM

similarity against collection X, as follows. (The $\dot{-}$ operator yields 0 if the difference is negative.)

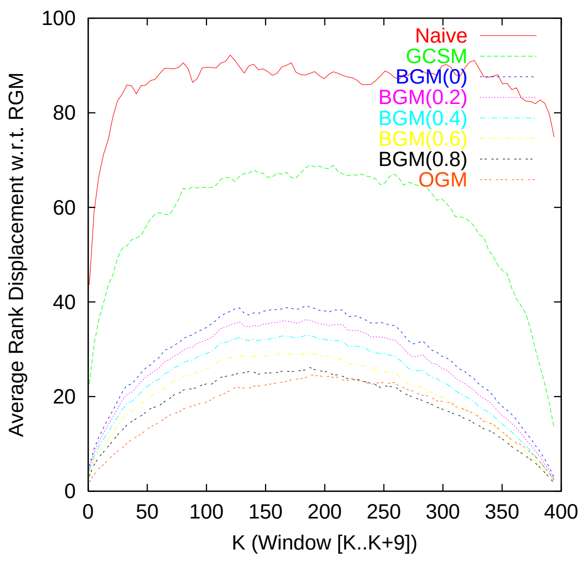
$$TopKDist_{M_1,M_2,K}(X) = \frac{\sum_{i=1}^{K} rank_{M_2,X}(L_{M_1}(X)[i]) \dot{-} i}{K}$$

We also define a distance measure over a "window" of the ranked lists. This definition is identical to the previous one, except that instead of looking at the Top K according to the first measure, we look at collections in a specific window. In this case, we cannot omit downward displacements, since omitting them would make windows in the lower segments of the list appear closer. For the example in Figure 7, the average distance for a window of size 3, starting at position 2, i.e., covering collections B, C and D, is given by $\frac{1+1+1}{3} = 1$. Formally,

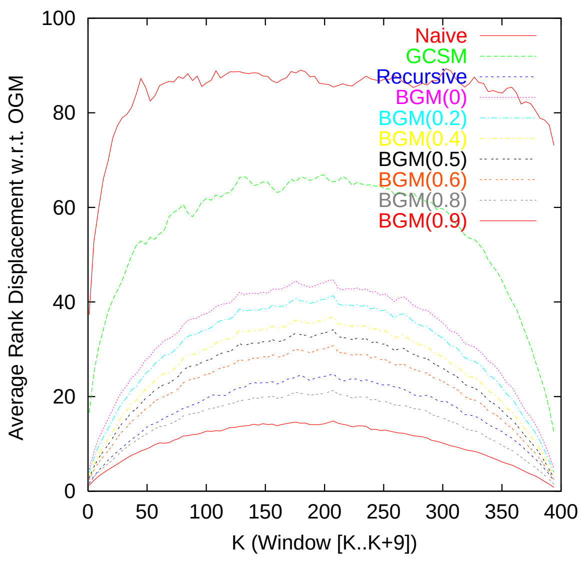
$$WindowDist_{M_1,M_2,I,K}(X) = \frac{\sum_{i=I}^{I+K-1} |rank_{M_2,X}(L_{M_1}(X)[i]) - i|}{K}$$

This measure helps us analyze how well two measures agree in different segments of the ranked lists that they produce. Notice that the ranked lists we have seen so far have been generated by picking an arbitrary collection X, and arranging all other collections by their similarity to it. Thus, in order to be able to compare two measures, we average the distances we compute over all possible choices for X.

Figure 8 shows the average rank displacement in the Top K list for various measures with respect to RGM, as a function of K. Notice that the average displacement for the Naive measure, even for the Top 10, is as much as 40, which is about 10% of the size of the entire corpus. This



(a)



(b)

Figure 9: $WindowDist_{10}$ w.r.t. (a) RGM and (b) OGM

result means, informally, that the collections that RGM considers the top 10 would be, on average, around the 40th or 50th position under the Naive measure, a very significant difference! On the other hand, all the Genealogy measures are bunched around the bottom of the graph, with even their peak displacement being well under 20. In fact, for the Top 10 list, the average displacement between RGM and OGM is just 1.42. This result means that RGM and OGM agree very well on what the most similar collections to a target collection X are. For the BGM family, as the value of β decreases, the displacement starts getting larger and larger, but it is still much smaller than the displacement of GCSM, and that of the Naive measure.

It is important to realize that we can only compare measures with respect to RGM from this graph. For example, the displacement between the GCSM and the Naive measure is *not* given by the difference between the curves corresponding to them on this graph. Also notice that all the curves have roughly the same shape, rising for a while before dropping off again. This behaviour is illustrated better by Figure 9(a), which shows the average rank displacement, in a sliding window of size 10, of all measures with respect to RGM.

The shape of the curve tell us that, for all the measures, there is greater agreement at the beginning and the end of the list than in the middle: there are a few collections which are clearly the most similar and there are a few collections which are clearly the most dissimilar. These collections are more easily identified by all the measures and, therefore, they agree more in the beginning and the end.

Figure 9(b) plots a similar graph, this time comparing the various measures to OGM. Once again, we notice that the Naive measure produces results that are extremely different from the results produced by OGM. For BGM, the distance from OGM gets larger as β decreases, which is only to be expected since OGM has $\beta = 1$. But the rankings appear much less sensitive to β at the beginning and the end of the ranked lists. We also see that the curve for RGM lies between the curves for $\beta = 0.6$ and $\beta = 0.8$. This does *not* mean that the RGM behaves like the BGM with $\beta = 0.7$. All it means is that RGM is as different from OGM as the BGM with $\beta = 0.7$.

These graphs conclusively establish that using a hierarchy makes a big difference to the similarity rankings that are generated. We also conclude that GCSM is rather different from the Genealogy measures, a fact that we attribute to GCSM's use of many-to-many matches. BGM is sensitive to the specific choice of β, but the sensitivity is much lower at the top and the bottom of the lists. Thus, the choice of β is not too critical, if one is trying to identify clearly similar or dissimilar collections. RGM and OGM are extremely similar at the top of the list, which is to be expected in this domain. Our data set does not have any multiplicity at the leaf level (there is multiplicity at higher levels), since it was rarely the case that a student repeated a course and ended up with the same grade.

5.2 Matching Human Intuition

In order to understand how well the various measures match human intuition, we performed an informal user study. Some of the important issues in the design of the study were:

- The users needed to be familiar with the domain from which the collections were drawn. With this in mind, we chose the supermarket domain, and each collection was a bag of grocery items.
- It was not reasonable to expect users to come up with absolute similarity scores between collections. Instead we asked users to rank two collections according to their similarity to a given collection.
- The collections needed to be reasonably small in order to keep the questions tractable. Therefore, we used collections with a small number of distinct elements. Fortunately, these collections proved sufficient to test the validity of the premises underlying the different measures.

The study was carried out on 33 people, all members of the Stanford Database Group. It consisted of 10 multiple-choice questions such as the ones shown in Figure 10. The study was

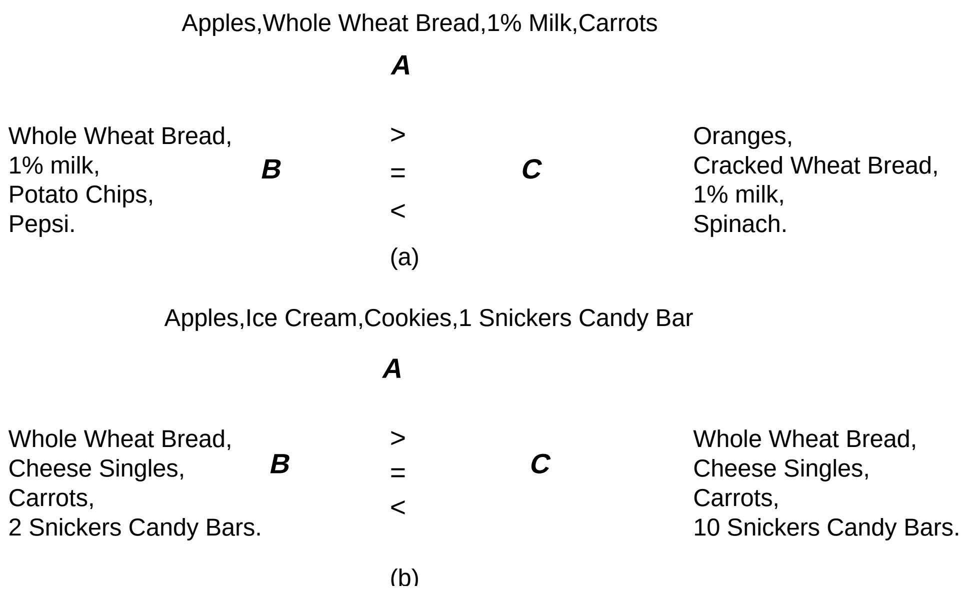


Figure 10: Two Sample Questions

designed such that the answer to each question would be a vote for or against one or more measures.

Figure 10(a) shows a sample question from our user study. It shows three collections A, B and C, and the user needs to determine whether $sim(A, B)$ is greater than, equal to, or less than, $sim(A, C)$. If the user thought that customer C was more similar to A (than B was to A), say because C and A appear to like health foods, then the user would circle the "<" symbol. In this example, A and B have two elements in common, while A and C have just one element in common. Thus, the traditional measures would report that $sim(A, B)$ is higher. On the other hand, 75% of the users decided that $sim(A, C)$ is higher, which agrees with all the First and Second Generation measures. We conclude that, in this case, the First and Second Generation measures perform better than the traditional measures.

Figure 10(b) shows another sample question from the survey. In this case, OGM would predict that $sim(A, C)$ is higher. BGM predicts that $sim(A, B)$ is higher (for most β), while RGM predicts that B and C are equally similar. Of the 33 users, 28 agreed with either BGM or RGM, from which we deduce that OGM does not match human intuition. The distribution of responses was not clear enough for us to choose between BGM and RGM in this case.

We do not have space to report all our results, but briefly, the following conclusions were drawn from the survey:

- Using the hierarchy is definitely an improvement over a naive approach, and more intuitive similarity results are obtained.

- GCSM does not perform as well as the Genealogy measures in this domain, and cannot be recommended as a general-purpose measure.
- There was a lot of support for both BGM and RGM, and the variance of the results ruled out our being able to decide whether one was better than the other. In practice, one would need application-specific experiments to determine which measure better matches the application semantics.
- For BGM, it was established that β values of 0 and 1 are both unsatisfactory. Again, application semantics would determine the exact value of β although the reasonably low sensitivity to β in our experiments in Section 5.1 suggests that a β value around 0.5 is reasonable.

While the user study was fairly limited in its scope, we believe that its conclusions are nonetheless useful and generally acceptable in most domains. We provide more details on the user study in the appendix.

6 Related Work

There have been attempts to improve traditional cosine similarity, as well as address data sparsity, using dimensionality-reduction techniques such as Latent Semantic Indexing [DDF$^+$90]. This technique actually shows some *improvement* in the quality of the similarity scores, since it tries to infer latent relationships between dimensions. Such techniques have also been tried in collaborative filtering [SKKR00] but it appears somewhat unclear as to whether it actually improves recommendation quality. Notice that using a domain hierarchy is actually an implicit form of dimension reduction, since the hierarchy implies that all elements are not orthogonal to each other. On the other hand, our techniques explicitly define the relationship between the different dimensions, while LSI infers the relationships from the corpus.

There have been quite a few attempts to use word hierarchies such as WordNet [MRF$^+$90] in Information Retrieval. Rada et al. [RMBB89] defined the semantic similarity between two words as the weight of the path between the words, which bears a lot of resemblance to our definition of the *LCA* of two leaves. Lee et al. and Kim et al. [LK93, KK90] have also used this "conceptual distance" measure for Information Retrieval. There are also other, information-based measures based on the same hierarchy [Res95] which can be used for word similarity. Richardson et al. [RS95] compare the efficacy of different word-similarity measures in computing query-document similarity. All these works are focussed on query-document similarity and do not generalize to inter-collection similarity.

Richardson et al. [RS95] also discuss the issue of generating edge weights for the concept graphs, which could find use in our work in generating edge weights for our hierarchy.

Scott et al. [SM98] have studied the use of a *hypernym density* representation instead of a bag-of-words representation in text classification and report improvements for corpora with a reasonable amount of diversity. Rodriguez et al. [dBRGHDA97] also report improvements in text classification when using WordNet to enhance neural-network learning algorithms. But neither of these works use a direct similarity measure based on the hierarchy. Concept hierarchies have frequently been used in data mining. They have been used to mine multi-level association rules [HF95, SA95], and to improve knowledge discovery in textual databases [FD95]. Neither of these two applications is directly related to computing similarity using hierarchies.

There are also other classes of methods used to compute similarity between collections which exploit the structure *between* collections. For example, [BLG98] uses the link structure of research papers to compute similarity between them. Such methods are not directly related to our work, except they may perhaps be used to solve the same overall problem.

Besides the similarity measures that we have described in Section 2, there have been a variety of distance measures defined in various contexts. One such class of measures is *edit distance*, wherein the distance between two structures is measured by the cost of the edit operations needed to transform one structure to the other. Algorithms for finding the optimal edit script exist for various types of structures, and for various sets of edit operations. Computing the optimal edit distance between unordered trees, even with simple edit operations, is NP-complete [SZ97]. In addition, edit distance does not give us the freedom to deal with nuances of inter-collection similarity, such as handling multiple occurrences.

Another distance measure, popular in a variety of domains, is *Earth-mover's distance* [RTG98], which measures the distance between two collections of points in space by calculating the work to be done in moving mounds of earth, located at points in the first collection, to fill holes, located at the points in the second collection. Once again, this model is not a good fit for the problem at hand because it forces many-to-many match semantics and, again, does not handle multiple occurrences well.

7 Conclusions

We have proposed exploiting hierarchical domain structure to compute similarity between collections. We defined measures that use this hierarchy, shown why both these and traditional measures often have unsatisfactory semantics, and suggested refinements that provide good semantics for inter-collection similarity. We have performed empirical comparisons of our measures with traditional similarity measures, and shown that using the hierarchy makes a large difference, both in terms of the values that are produced, and in terms of ranked lists of collections similar to a given collection. We have reported the findings of an informal user study to justify our belief that our measures generate results that are closer to human intuition than the traditional similarity measures.

We are currently in the process of building recommender systems using these measures, and using the hierarchy in other portions of the recommender system. Preliminary results are encouraging, and seem to provide higher-quality recommendations than the simple Pearson-correlation-based, nearest-neighbor approaches.

References

[BHK98] John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proc. 14th Annual Conference on Uncertainty in Artificial Intelligence*, 1998.

[BLG98] Kurt Bollacker, Steve Lawrence, and C. Lee Giles. CiteSeer: An autonomous web agent for automatic retrieval and identification of interesting publications. In *Proc. 2nd International Conference on Autonomous Agents*, pages 116–123. ACM Press, 1998.

[dBRGHDA97] Manuel de Buenaga Rodríguez, José Maria Gómez-Hidalgo, and Belén Díaz-Agudo. Using WordNet to complement training information in text categorization. In *Proc. 2nd International Conference on Recent Advances in Natural Language Processing*, Tzigov Chark, BL, 1997.

[DDF$^+$90] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic indexing. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

[FD95] R. Feldman and I. Dagan. Knowledge discovery in textual databases. In *Proc. KDD-95*, 1995.

[GNOT92] David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.

[GSK$^+$99] Nathaniel Good, J. Ben Schafer, Joseph A. Konstan, Al Borchers, Badrul M. Sarwar, Jonathan L. Herlocker, and John Riedl. Combining collaborative filtering with personal agents for better recommendations. In *AAAI/IAAI*, 1999.

[HF95] J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. In *Proc. VLDB '95*, pages 420–431, 1995.

[KK90] Y. Kim and J. Kim. A model of knowledge based information retrieval with hierarchical concept graph. *Journal of Documentation*, 46(2):113–116, 1990.

[LK93] J. Lee and M. Kim. Information retrieval based on a conceptual distance in is-a heirarchy. *Journal of Documentation*, 49(2):188–207, 1993.

[McG83] M. J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, 1983.

[MRF$^+$90] G.A. Miller, Beckwith R., C. Fellbaum, D. Gross, and K.J. Miller. Introduction to wordnet: An on-line lexical database. *Journal of Lexicography*, 3(4):234–244, 1990.

[Nap] Napster. http://napster.com.

[Ope] The Open Directory Project. http://dmoz.org/.

[Res95] Philip Resnick. Using information content to evaluate semantic similarity in a taxonomy. In *IJCAI*, pages 448–453, 1995.

[RIS$^+$94] P. Resnick, N. Iacovou, M. Sushak, P. Bergstrom, and J. Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *Proc. Computer Supported Collaborative Work Conference*, 1994.

[RMBB89] R. Rada, H. Mili, E. Bicknell, and M. Blettner. Development and application of a metric on semantic nets. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(1), 1989.

[RS95] Ray Richardson and Alan F. Smeaton. Using WordNet in a knowledge-based approach to information retrieval. In *Proc. 17th BCS-IRSG Colloquium on Information Retrieval*, 1995.

[RTG98] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. The earth mover's distance as a metric for image retrieval. Technical Report STAN-CS-TN-98-86, Stanford University, 1998.

[SA95] Ramakrishnan Srikant and Rakesh Agrawal. Mining generalized association rules. In *The VLDB Journal*, 1995.

[SB88] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523, 1988.

[SGM95] N. Shivakumar and H. Garcia-Molina. Scam: A copy detection mechanism for digital documents. In *2nd International Conference in Theory and Practice of Digital Libraries*, 1995.

[SGM00] A. Strehl, J. Ghosh, and R. Mooney. Impact of similarity measures on web-page clustering. In *Proc. AAAI Workshop on AI for Web Search*, 2000.

[SKB$^+$98] Badrul M. Sarwar, Joseph A. Konstan, Al Borchers, Jonathan L. Herlocker, Bradley N. Miller, and John Riedl. Using filtering agents to improve prediction quality in the grouplens research collaborative filtering system. In *Computer Supported Cooperative Work*, 1998.

[SKKR00] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Application of dimensionality reduction in recommender system–a case study. In *ACM WebKDD 2000 Workshop*, 2000.

[SKKR01] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proc. 10th International WWW Conference*, 2001.

[SM98] Sam Scott and Stan Matwin. Text classification using WordNet hypernyms. In *Proc. Use of WordNet in Natural Language Processing Systems*. Association for Computational Linguistics, 1998.

[SZ97] Dennis Shasha and Kaizhong Zhang. Approximate tree pattern matching. In *Pattern Matching Algorithms*. Oxford University Press, 1997.

[van79] C. J. van Rijsbergen. *Information Retrieval.* 2nd ed., Butterworths, 1979.

Appendix A The User Study

The user study that we undertook is shown below. Nearly all the questions had more than 70% of the users selecting one of the three choices. If this happened, the answer was considered to be the "correct" answer. The questions were designed in such a fashion as to make the different similarity measures provide different answers. Thus, identifying the "correct" answer to a question would help us decide that some similarity measures are better than others.

The only questions for which we could not determine a "correct" answer were designed to choose between BGM and RGM. Here, user votes were split between $\approx$ and one of $>$ or $<$. For example, Question 5 had votes split between $\approx$ (which is the answer suggested by RGM) and $>$ (which is suggested by BGM). We would have needed a more precise characterization of the similarities involved, perhaps in terms of exact numerical values, in order to be able to draw conclusions on the relative quality of RGM and BGM in these cases.

1. { Diapers, Fat-free Milk, Gerber Squash and Corn }

Baby Bee Apricot Baby Oil	$>$	Fat-free Milk
Gerber Green Beans	$\approx$	Diet Pepsi 6Pk.
1% Milk	$<$	Equal low calorie Sweetener
Miller Lite 6Pk.		

2. {Apples, Ice Cream, Cookies, 1 Snickers Candy Bar }

Whole Wheat Bread	$>$	Whole Wheat Bread
Cheese Singles	$\approx$	Cheese Singles
Carrots	$<$	Carrots
1 Snickers Candy Bar		2 Snickers Candy Bars

3. {Apples, Ice Cream, Cookies, 1 Snickers Candy Bar }

Whole Wheat Bread	$>$	Whole Wheat Bread
Cheese Singles	$\approx$	Cheese Singles
Carrots	$<$	Carrots
2 Snickers Candy Bars		10 Snickers Candy Bars

4. { 1 gallon 1% milk }

1 gallon 1% milk	>	1 gallon 1% milk
1 gallon 2% milk	≈	1 gallon Orange Juice
	<	

5. { Apples, Ice Cream, Cookies, 1 Snickers Candy Bar }

Whole Wheat Bread	>	Whole Wheat Bread
Cheese Singles	≈	Cheese Singles
Carrots	<	Carrots
20 Snickers Candy Bars		40 Snickers Candy Bars

6. { Whole Wheat Bread, 1% Milk, Cookies }

Whole Wheat Bread	>	Whole Wheat Bread
2% milk	≈	2% milk
1 Snickers Candy Bar	<	2 Snickers Candy Bars

7. { Whole Wheat Bread, 1% Milk, Cookies }

Whole Wheat Bread	>	Whole Wheat Bread
2% milk	≈	2% milk
2 Snickers Candy Bars	<	10 Snickers Candy Bars

8. { Apples, Whole Wheat Bread, 1% Milk, Carrots }

Whole Wheat Bread	>	Oranges
1% milk	≈	Cracked Wheat Bread
Potato Chips	<	1% milk
Pepsi		Spinach

9. { Whole Wheat Bread, 1% Milk, Cookies }

Whole Wheat Bread	>	Whole Wheat Bread
2% milk	≈	2% milk
10 Snickers Candy Bars	<	20 Snickers Candy Bars

10. { 2 gallons 1% Milk }

1 gallon 1% Milk	>	1 gallon 1% Milk
1 gallon 2% Milk	≈	1 gallon Orange Juice
	<	