The Academy:
A Community of Information Retrieval Agents

Robert France
Draft 0.1
6 September 1994

Introduction and Motivation

0.1 The Document Universe

[[It is very large. It is dynamic.]]

[[It is not just made up of documents. Other important classes of entities include people – as authors, editors, illustrators – institutions – as authors, sponsors, work places for people – and controlled concept systems – flat, hierarchical, and network.]]

0.2 Text, Structured Text and Hypertext

We commonly picture text as a sequence of words; or alternatively as a sequence of paragraphs, each of which is composed of a sequence of sentences, each of which is itself a sequence of words. It is worth noting, however, that even in the most impoverished text production systems, this has never been completely true. Formatting information. even if only in the form of explicit character returns and spaces, has always been used and been significant. ... With the advent of mark-up languages and with the proliferation of word processors and desk-top publishing software, ...

It is also worth noting that text is not so much a sequence of *words* as a sequence of *terms*, including most commonly words, but also including names, numbers, code sequences, and a variety of other $#*&)&@^ tokens.

Just as we commonly simplify text into a sequence of words, so too it is common in information retrieval to regard documents as single texts. Nothing is less common, though, than a document with only a single part, and that unstructured text. Typical documents have titles, authors, abstracts, distribution lists, synopses, subject lines, lists of

keywords, and/or any of a host of dates.

The natural extension of this concept is to the universe of hypertext, where texts, structured texts, and compound documents are linked together by relationships including ... . Search and retrieval in such a universe involves new questions: Where does a document begin and end? How can we decide how much to show to a user? When does a query need to be matched by a single node in a hypertext, and when may partial matches in several nodes count?

0.3 Approximate Matching

One way in which information retrieval systems tend to differ from classical database systems is that the criteria for match of a text or a document to a query tend to be approximate. ... Ranked retrieval ... Boolean systems & extended Boolean systems

1 Foundations

1.1 Objects, Weighted Objects, and Weighted Object Sets

```
//    DESCRIPTION:  A weighted object set is a subclass of set that makes
//          use of the information that its elements are weighted objects.
//          Like any good set class, this class provides methods for
//          testing for the inclusion of elements (isElt(), but note the
//          extra argument) and for iterating through the set.  The
//          unique aspect of the construct is that the weights on elements
//          provide an order.  In particular, one iterates through
//          the set in non-decreasing order by weight.  This strong
//          ordering encourages a skipToKth() function, allowing one to
//          start the iteration somewhere else than on the first element,
//          and the sample() functions, which return groups of objects.
//          In particular, sample() returns (up to) a requested number
//          of weighted objects starting from a particular point in the
//          iteration, and sampleToWt() all the weighted objects from
//          a particular point down to a minimum weight.
```

1.2 Basic Weighted Object Sets

[[Posting lists and posting sets.]]
[[Application: term inversion sets & Zipf's law]]
[[Singleton wtdObjSets]]

1.3 Weighted Object Set Operations

Weighted object sets support the same operations as do classical sets: intersection, union, difference, cross product, and so forth. The intersection of two wtdObjSets, for instance, is defined as the weighted object set made up of all the members that belong to both sets; the union, the wtdObjSet made up of all the members of either. The problem with defining these operations comes in determining the weight function in the resulting set. Suppose we have two sets, each of which contain object **a**, the first at weight **w** and the second at weight **v**. Then both the intersection and union of the two sets contain object **a**, but at what weight? **w**, **v**, or some other weight entirely?

[[The weight-combining function is independent of the operator, although certain functions will prove more appropriate to certain operators ...]] Some functions make better sense – or as we say in computational linguistics, have better semantics – in some operations; others in others. We have focussed on a few combinations that we suspect of being fruitful. In the end, any functions and operations must be evaluated in context outside of information retrieval for their validity and utility. [[formal; user studies.]]

[[maximum]]

[[summative functions:

average & weighted average
vector cosine for some direct product
other vector space similarity functions: Croft, Harmon
sigmoid
relative entropy]]

[[quorum functions]]

The cross product operator implies the existence of tuples, and ultimately relations and functions within the category of weighted objects. These important representation tools are finding a place in document retrieval in the context of hypertext.

[[link projection searcher]]

2 Implementation Issues

2.1 Lazy Evaluation

Searchers of the Academy are all lazy in evaluating set order of the combined set. The difference between an explicitly opportunistic searcher like Occam and a purportedly exhaustive searcher like Aquinas is in evaluation of the final object weights. These weights are determined during the process of running elements from the component sets into the accumulator bank, producing set images in the form of linked lists threading the accumulator bank. In varying proportion, some elements fall into the intersection of the component sets, others belong to only a single component.

Elements that belong to only a single component set are generally the lowest weight in the combined set, and thus the least likely to ever be sampled. The Academy searchers

conserve cycles by not attempting to put these elements into a merged order until they are asked for. Instead, elements that occur only in a given component set are kept in a "set image" list until they are asked for. Since this image is built in the order in which elements are drawn from the component wtdObjSet, it is itself in order.

2.2 Probing

2.3 Enabling Data Structures

A searcher is a device for combining weighted object sets. The primary data structure for this operation, the holding structure for set combined set contents, is the accumulator bank. The secondary data structure is the collection of weighted object sets being combined.

2.3.1 Weighted Object Set Collection

The first structure involved in an Academy searcher is the weighted object set collection. Exactly as one might expect from its name, this is a structure for keeping together the collection of component sets that will be combined in the search operation. Its primary function is to allow a single iteration through the elements of the component sets, including duplicates in IDs where objects occur in more than one set, but ensuring a wtdObjSet-like non-increasing sequence through all the objects in the collection.

> **Implementation Note:** The iteration operators for a weighted object set make use of Pixes that are converted as needed into weighted objects. Among other motivations, this system allows more than one client to iterate through a wtdObjSet at a time. A weighted object set collection, on the other hand, is designed to have one and only one client: the searcher that makes use of it. Thus weighted object set collection iteration operators return weighted objects, and the extent to which the collection has been explored is maintained in the collection as part of the object state.

The other functions of a weighted object set are statistical in nature, and calculate such useful values as the maximum and expected number of unique elements left in the unexplored portion of the collection, the expected weight of an unseen element, and the number of elements likely with weight higher than some threshold. In this section, we develop the first of these.

Consider first a collection of two weighted object sets, regarded as two samples of the total object space. We can calculate the expected number of unique elements in the combined set as:

size of set #1 + set of set #2 - size of overlap.

The size of the overlap is of course a random variable. As with all the collection

statistical functions, there are two values of this variable that are of interest to the searcher: the mean value, and the maximum value within a certain window. For instance, in computing the approximate size of the union of our component sets, we would be interested in the mean. In opportunistic search, on the other hand, we are interested in the maximum value that can reasonably be expected. We sharpen "can reasonably be expected" to mean "occurs with a probability greater than some small value." Since an overlap of zero happens very rarely, this can be determined

We can derive the size of the overlap using the hypergeometric distribution, which governs such cases. The probability distribution for the hypergeometric distribution is a relatively complex formula, but its mean is the same as the mean of the binomial distribution. If we label the sizes of sets 1 and 2 by n1 and n2 respectively, and the size of the object class (the total number of objects from which the sets are drawn) as N, the mean overlap is

$$n1 \cdot n2 \;/\; N$$

Thus for two sets, the mean size of the union is

$$n1 + n2 - n1 \cdot n2/N.$$

For k sets of sizes n1, ..., nk, this formula becomes

$$\sum_{i=1}^{k} (-1)^{i-1} \cdot \frac{\sum \text{(all combinations of the nj taken i at a time)}}{N^{i-1}}$$

In the case where k=4, for instance, the formula is:

$$n1 + n2 + n3 + n4 - (n1 \cdot n2 + n1 \cdot n3 + n1 \cdot n4 + n2 \cdot n3 + n2 \cdot n4 + n3 \cdot n4) / N + (n1 \cdot n2 \cdot n3 + n1 \cdot n2 \cdot n4 + n1 \cdot n3 \cdot n4 + n2 \cdot n3 \cdot n4) / N^2 + n1 \cdot n2 \cdot n3 \cdot n4 / N^3$$

In the case where all the nk are equal, the formula is the more familiar:

$$\sum_{i=1}^{k} (-1)^{i-1} \cdot \text{binom}(k, i) \cdot \frac{n^i}{N^{i-1}}$$

Unfortunately for our analysis, this situation hardly ever occurs in practice. However, when N is large compared to the ni, which it usually is in text retrieval, N^{i-1} will come to dominate the sums in the numerators as i increases, and we can ignore the later terms in the summation.

```
//      expectedEltWt(), expectedNumElts()
//
//      PURPOSE:  Statistical functions returning the expected total weight
//              of an element in the collection (i.e., the total weight from
//              all the sets in the Image cells), and the expected number of
//              unique elements in the collection.
//
//      FORMULAE:  The expected number of unique elements in the collection
```

```
//          is the total number across all sets less the amount of overlap
//          between the sets:
//                N * (1 - (1-f1)(1-f2)...(1-fk)) =
//                   (N**k - (N-n1)(N-n2)...(N-nk))
//          where N is the total number of objects in the element class and
//                fi = ni / N
//          is the proportion of those objects in set i.
//            The expected weight of a hit is somewhat more complicated.
//          If all the ni were equal to some n, then the number of elements
//          that were hit j times (ocurred in j componenet sets) would be
//                (j k) (f**j)*((1-f)**(k-j))
//          where (j k) is the jth binomial coefficient in a k-adic
//          sequence, k being here the number of lists.  So for 4 lists
//          the proportion of elements in the class to recieve hits is:
//                0 hits:        (1-f)**4
//                1 hit:        4f(1-f)**3
//                2 hits:       6(f**2)(1-f)**2
//                3 hits:       4(f**3)(1-f)
//                4 hits:        f**4
//          The expected number of hits is thus the weighte sum of the
//          sequence, in this case:
//             4f(1-f)**3 + 2*6(f**2)(1-f)**2 + 3*4(f**3)(1-f) + 4*f**4
//          Trust me; if you do the algebra this reduces to:
//                4f
//          and not just for the case where k=4, but for any k.  So for
//          k sets all of the same size n in a universe of N objects,
//          where each element in any component set has weight 1, the
//          expected weight on an element in the combined set is:
//                N * k * f * 1 =  k*n
//          It remains to generalize this to the real case, where the
//          ni and thus the ki are different, and where element weights
//          in the component sets are not all 1.  I haven't done the
//          algebra on this, but my suspicion is that:
//                (n1*w1 + n2*w2 + ... + nk*wk)
//          where wi is the expected weight of an element in set i is
//          certainly close, and may even be exactly right.
//            Of course, we don't really know the expected weight for
//          an element in one of our component sets, especially after
//          the set has been sampled repeatedly.  So we use the weight
//          of the top element, which is at least a max.
//            What this figure gives us is the total weight over the
//          entire universe.  We can use this figure directly to
//          calculate the expected weight increase resulting from a hit
//          onto a segment of the accumBank, or we can divide it by the
//          number of possible events (the size of the object class; the
//          number of possible microstates) or the number of unique
```

```
//          elements in the setImageColl to discover various kinds of
//          averages.
//
//---------------------------------------------------------------------
```

2.3.2 Accumulator Bank

In many applications for combining weighted object sets, there is need to combine weights derived from many sources but related to a single object. This function is served by an accumulator bank. An accumulator is an extension of a weight: it is a data type that includes all possible weights, but may also include other values. For instance, in MARIAN, weights are represented by real numbers between 0 and 1. This representation has the advantage of providing a top element, used to signify a perfect match, but it has the disadvantage that it is not closed under addition. Adding two weights together may produce a value greater than one. Since such additions are part of the intermediate calculations of new weights, we can represent an accumulator by a non-negative real number. Other systems make use of short integers for weights; for these systems, the obvious choice for an accumulator is a long integer.

An accumulator bank is a table of accumulators accessed randomly by object (usually document) ID. In the Academy, the random-access structure is provided by subclasses of the weighted object table (q.v.). These tables are extensible hash tables of cells, each of which contains an extended weight type and a fullID, and each of which can be accesses in linear time by fullID. The subclasses used by the Academy searchers also contain further structure in the form of list threads.

All of these searchers make use of some variation on the threaded accumulator bank. An accumulator bank (see Harmon in Frakes et al.) is a table of weights ("accumulators") accessed by object ID and used to accumulate the weights of wtdObjs in the wtdObjSets being merged. All the accumulator banks that we use are maintained in some order through being "threaded" by doubly linked lists. How these lists are constructed and how many there are vary between searchers.

Each searcher in the Academy maintains a single accumulator bank. In all cases, the bank is threaded by at least two categories of lists: single-hit lists and multiple-hit lists. The single-hit list or lists, as the name implies, contain wtdObjs that have been found in only a single component set. These lists are invariably maintained in non-increasing order by weight: since the component sets are generally being explored in that order, this is never very expensive. Multiple-hit lists vary more widely: there may be one or many per searcher, and some are maintained in a partial (heap) order.

A cosine searcher uses only a single multiple-hit heap, called "Violations" in the object definition because a second hit on any element usually induces a violation in the threading order. WtdObjs that have been so hit and thus moved to the Violations heap need to be re-inserted in the main order before the searcher can be said to be stable.

A quorum searcher that is merging k wtdObjSets maintains (k-1) multiple-hit lists. These lists are sorted and returned in order of decreasing number of hits so that the first

elements drawn from the union set are those which appear in the maximal number of component sets.

An exhaustive searcher merging k sets maintains k lists in the single-hit category; each the image of one set. Most of the items in these lists will never be returned to the user of the searcher, although some of the highest-weighted may. In addition, the searcher maintains a "completed" list of items in globally non-increasing order. As the merged set is explored by its user, items are detached from the top of the k component lists and the Violations heap(s) to extend this completed list.

An opportunistic searcher has a single list threading through the single-hit category, together with a pointer into the list at a point above which it is deemed to be stable by the statistical "stopping rules" that make the searcher opportunistic.

2.3.3 Set Image and Set Image Collection

> **Implementation Note:** There is an intimate relationship between wtdObjSetColls and setImageColls; so much that the two classes use [[the same/ sibling]] class[[s]] as cells. A wtdObjSetColl can itself serve as the [[composite]] image of the component sets in the searcher. Maximizing searchers can work this way, as they only need to examine elements in weight order. More commonly, the wtdObjSetColl is transformed into a setImageColl; either all at once in exhaustive summation searchers, gradually in opportunistic summation searchers, or set by set in quorum searchers. This process is performed by manipulating and reordering the set image elements (setImageCollCell) without transforming them. Quorum and exhaustive summation searchers pass the cells from the wtdObjSetColl to the SetImageColl when the searcher is finalized. Opportunistic searchers [[keep the cells in both wtdObjSetColl and setImageColl linked lists at the same time]] [[[actually, we might could move them all to the setImage, if we can avoid using transferNum() and implement only transferToWt()]]], moving individual weighted objects from wtdObjSet to setImage as needed.

This structure is designed to be used in searchers together with some sort of "accumulator bank." The searcher functions to create a single combined wtdObjSet from the several sets in the wtdObjSetColl by iteratively sampling the wtdObjSetColl and inserting the wtdObjs found into the accumulator. This intended use drives several aspects of the design.

First, it is assumed that a given wtdObjSetColl will have only one user. The individual sets in the collection may be shared -- with other collections, most likely -- but the collection itself belongs to a single searcher and will only be used by that searcher. This means that state information can be kept locally where it is handy. In particular, the Pixes that denote how far each component set has been explored are kept in the collection structure. Also, it will be noted that the firstSample*() and nextSample*() methods do

not make use of Pixes to denote how far the collection has been explored. That information is implicit in the internals of the collection.

Second, we do not attempt to produce a true set of wtdObjs in the sampling methods. That is the function of the accumulator bank(s) within the searcher. The wtdObjBags produced in the sampling methods are truly bags.

On the other hand, each bag is produced in strict non- increasing order by weight, despite the added cost this entails. This decision is driven by the design of the accumulator banks. We want the one-hit bank of the searcher to be maintained in weight order, so that high-weight objects in that bank can be easily integrated into the samples that the searcher returns. Each wtdObjSet in a collection is in weight order, and at any point in the search process no unseen element remaining in the collection will have a higher weight than any in the one-hit bank. The task thus devolves to making sure that each sample is sorted before it is hooked to the end of the one-hit bank.

There are several approaches to doing this, but they can be divided into two categories: sort the collection sample bag after it has been produced, or produce it in sorted form. The first requires b*lg(b) operations, where b is the size of the bag. The second requires b*lg(k) operations, where k is the number of sets in the collection. Since we expect k to always be small compared to the average b, we choose to do the latter. (In actuality, neither of these worst-case bounds is likely, since large blocks of the wtdObjs in the sets and the resultant bag have equal weight. But the analysis holds just as well in this case: the number of times we must find either the next insertion point in the former case or the next set to sample in the latter is smaller, but the proportionate cost (lg(b) / lg(k)) is the same.)

IMPLEMENTATION NOTES: With all that said and done, we are beginning with the component sets in a sorted linear list, so that each set switch costs O(k) rather than O(lg(k)). The lg(k) heap (or binary sort) will come later.

3 Members of the Academy – Summative Union Searchers

A searcher is a mechanism for combining wtdObjSets into a single wtdObjSet. Any mechanism for implementing wtdObjSet intersection or union, for instance, would count as a searcher. The searchers in this collection (“the Academy”) implement particular union operations: those which weight the resulting merged set using cosine and quorum similarity functions.

Searchers differ in algorithm as well. Academy searchers use either exhaustive or opportunistic algorithms, soon to be supplemented by probing. These three choices: cosine or quorum, exhaustive or opportunistic, and probing or no probing, produce the basic eight searchers in the Academy.

3.2 AQUINAS

Exhaustive cosine searcher over a collection of weighted object sets. This is the "Aquinas" searcher, which processes all available information before returning any elements, but only sorts samples as needed.

3.1 OCCAM

The OCCAM searcher differs from Aquinas in that it attempts to examine only the information needed to establish the top portion of the combined weighted object set. Because of the structure of weighted object sets, it is often the case that no more than the top few elements of potentially huge sets are actually used. This is particularly true when the set ids being presented directly to a user, as humans find it irritating to look through more than a few hundred documents at a time.

Thus, once the top segment of the combined set can be established it is unnecessary to continue refining elements in the lower segment. This is the intuition underlying the design of Occam*.

> ***Footnote:** The acronym OCCAM stands for "Opportunistic Cartesian Cosine Approximate Matcher." As it stands, Cartesian cosine is only one of several summative functions that Occam can implement. But the acronym is nonetheless irresistible.

3.1.1 Opportunistic Search and "Stopping Rules"

We know – or could know if we bothered to measure a few parameters – the **drop-off function** for a basic weighted object set. And we can make informed guesses of the drop-off functions of other classes of weighted object sets. Even when we do not know the function and have no information about the wtdObjSet we can assume a worst-case scenario: that the set is flat and that all the elements have the same weight as the first element, or the deepest element that we have examined in the set. Since we are guaranteed to iterate through the set in non-decreasing order, the last element that we have seen provides a guaranteed maximum for weights in the unseen portion of the set.

> **Sidebar:** The drop-off [[attenuation]] function of a wtdObjSet is a statistical idealization of the actual sequence of weights found by iterating through the set. The drop-off function differs from the actual generated sequence in that it is smooth, continuous, and expressed by a relatively simple formula. The intention, as with any statistical idealization, is that over a sufficiently large number

of samples, the behavior of the actual enumeration would approach the ideal function. We will represent the drop-off function for a set **a** as $X_{\mathbf{a}}$.

Given a collection of wtdObjSets W_i with associated drop-off functions $X_{\mathbf{i}}$ ($1 \le i \le k$ for some k) we can calculate the drop-off function of the entire collection. Actually, we can calculate several functions. There is a family of drop-off functions for the sets formed by various set operations on the collection: $X_{\mathbf{U_{cos(W)}}}$, for instance, is the drop-off function of the summative union of all the W_i using a cosine combining function [[for common elements]]. Of more interest to the opportunistic search algorithm is the "uncombined" function $X_{\mathbf{W}}$, defined over the bag of elements obtained by iterating through the weighted object set collection.

We know rather a lot about this bag, and about its attenuation function. We know how many elements are in the bag: as many as are in any of the sets. We know the maximum weight of any element in the bag, and if we are permitted to know the minimum weights in the component sets, we also know the minimum weight in the bag. And if all the Ci have the same shape, which is certainly the case when we are combining term inversion sets, we can usually predict the shape of $X_{\mathbf{W}}$. For instance, if all the sets have constant drop-off functions, actual enumeration of bag weights will be a stairstep function. We can use a variety of smooth and continuous functions to approximate this, depending on the size of the component sets and their constant weights. In the case of combining term inversion sets with IDF weights, for instance, the collection drop-off function is generally [[hyperbolic, mirroring the Zipf curve that informed the weights, but is almost]] linear except in the head.

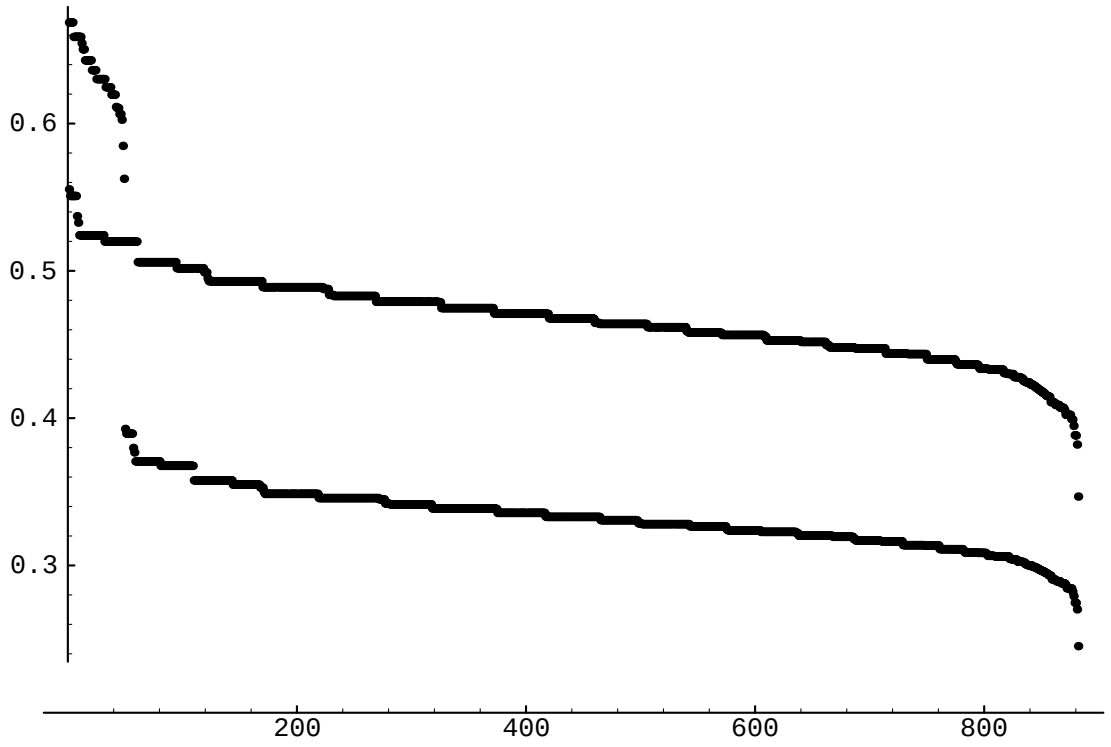

Fig. ATTENUATE: Actual attenuation functions for a typical union set using two different weight combining functions. This 882-element set is the union of the posting lists for "cat" and "dog" in the VT Library title collection. The central enumeration is for a maximized union. The function for a cosine summative union begins above the maximized enumeration, then drops below it when it reaches the end of the titles that include both words. Note that both are clearly stairstep functions.

Any drop-off function is to some degree an approximation of the actual enumeration of set or bag weights. The constant function that always returns the first weight in a collection is an approximation, albeit a poor one. A linear function from the highest weight to the lowest is also an allowable approximation. Where the size of the collection bag is large, and the component sets are in some sense a representative selection of the sets in the entire document collection, we can regard the collection bag as a random sample from the complete space of term-text association weights. We can construct the drop-off function of this space from empirical measurements. For instance, Fig. CORP.DIST shows the distribution of term-text weights for the corporate author name collection in MARIAN. Fig. CORP.ENUM shows the weight enumeration generated by this collection, and Fig. CORP.DROP.OFF a "best-fit"curve for the enumeration. When we are working with a collection of term inversion sets for corporate authors that is sufficiently large and well-distributed, we can use this function as a good approximation of the drop-off function $X_{\mathbf{W}}$.

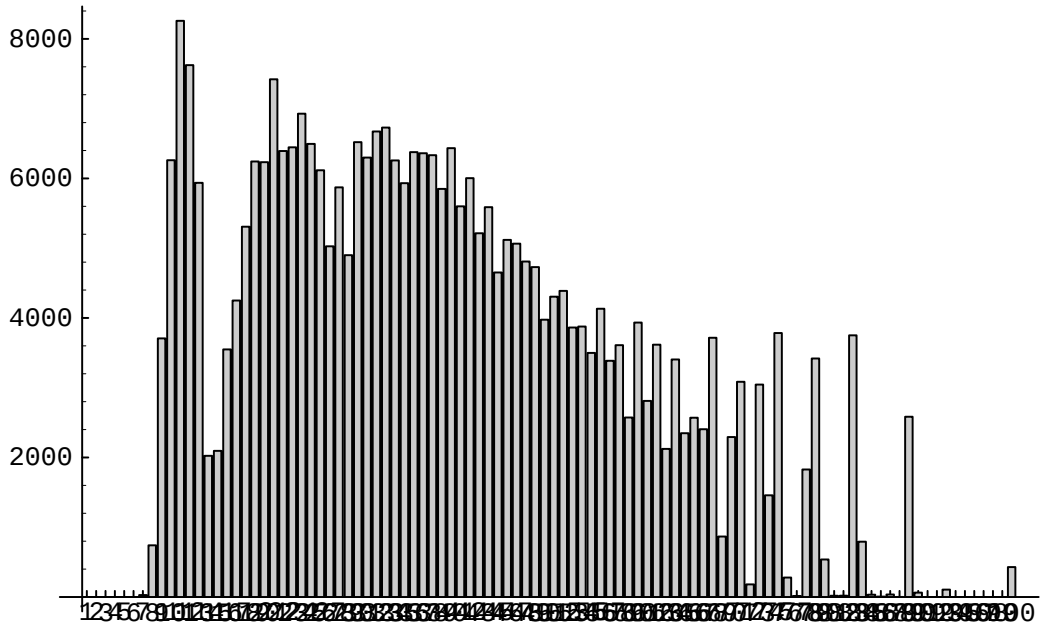

Fig. CORP.DIST: Number of term-text links in the collection of corporate authors in MARIAN. Each bar in the graph shows the number of links in a 1% window of the weight interval [0, 1].

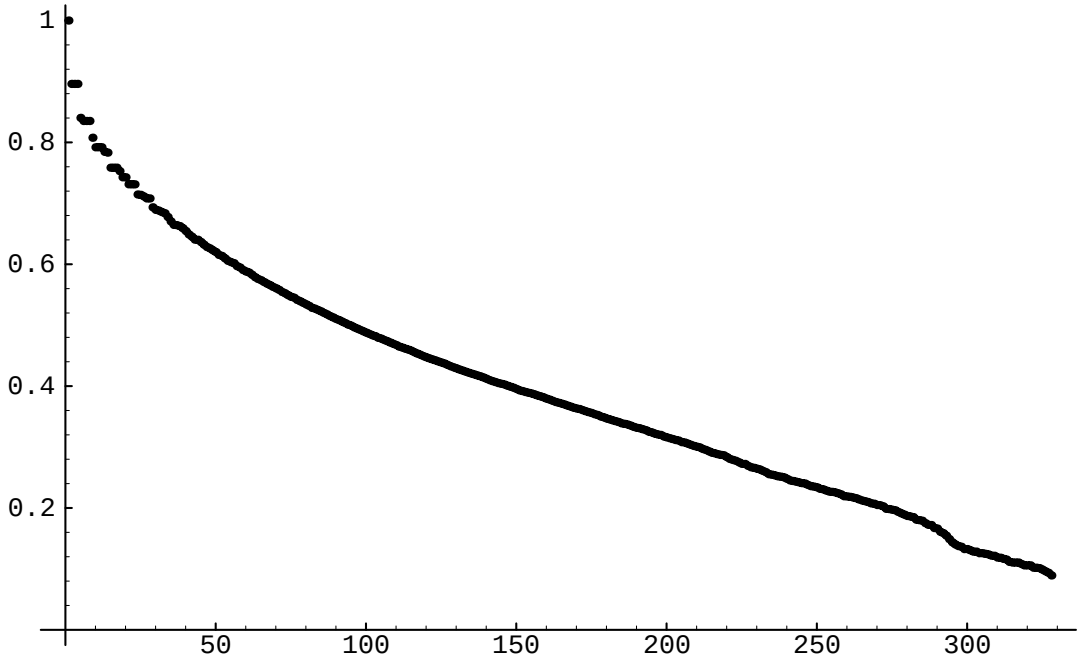

Fig. CORP.ENUM: Actual enumeration of term-text association values (posting list weights) for the corporate name collection.

Fig. CORP.DROP.OFF: The function that best captures the actual enumeration is ...

The possibility for opportunistic search occurs during a summative set operation over a large weighted object set, wither one that includes a great number of component sets or one that includes a few very large component sets, usually in addition to some shorter sets. Suppose that the search has been advanced to the point where some portion of the objects in the collection has been examined and stored in the accumulator bank. Let us simplify things a little by pretending that the ordering within the accumulator bank has been resolved, so that the bank presents us with a single weighted object set. In some sense, this set is an approximation of the set being constructed by the searcher. We imagine that the important part of this set is the head, either because the searcher is implementing an intersection operation and only those elements with a certain number of hits will end up in the set, or because it is implementing a summative union and all the elements are important, but only a sample off the top are being requested by the searcher's client.

We can now pose the question: what are the chances that exploring the rest of the way through the wtdObjSet collection will affect the top portion of the composite set? This can be broken into two question: what is the probability that one or more elements discovered in the additional exploration will change the order of elements in the top portion, and what is the probability that one or more elements in the remainder of the collection will change the composition of the top portion, either by accruing a total weight that belongs in the top portion or by promoting an element from the lower portion to the upper?

The first question can be re-cast simply as a question of sampling. Consider the top n elements of the accumulator to be a sample (of size n) drawn without replacement from the total class. In this case the class consists of all the objects that could be members of the combined weighted object set: all the documents, or titles, or names in the collection being searched. Call that total number N. The unexplored remainder of the wtdObjSet collection is a sample (of size m) drawn from the same collection *with* replacement. We are interested in the probability that these two samples have an element in common.*

> ***Footnote:** To be completely precise, we are interested in the probability that these two samples have an element in common where the total weight of the element in the weighted object set collection is greater than the difference in weight between the element in the accumulator and the element immediately above it in weight ordering: the probability that we will get a hit that displaces the element hit. In point of fact, however, the weight differences between neighboring accumulator cells in an accumulator bank of any size tend to be miniscule. Often there is no difference between neighboring elements, due to the stairstep nature of weighted object set enumerations. So we are not far off in asserting that *any* hit in an accumulator bank will displace the element hit.

If both samples were drawn without replacement, we would know the exact probability that they had at least one element in common. The probability that two

samples of size n_1 nd n_2 are disjoint is:

$$(N- n_1)!\cdot(N-n_2)! \ / \ N!\cdot(N- n_1-n_2)!$$

and so the probability of a hit is

$$1 - (N- n_1)!\cdot(N-n_2)! \ / \ N!\cdot(N- n_1-n_2)!$$

We cannot apply this formula immediately, since the sample from the wtdObjSet collection is drawn with replacement. In other words, we would need know the number of unique elements in the sample. Fortunately, we do know how the sample was constructed: it is composed of a certain number of samples drawn without replacement from the universal class. These samples are of course, the remaining unexhausted component sets in the collection: we know how many of these there are and how many elements in each remain unseen. Thus size of the total sample will therefore be a member of an extended hypergeometric distribution. [[explain in sidebar?]] We can use the distribution to calculate an expected size, or a maximum size within a given margin of error. For the purpose of this analysis, though, we will not be far off in simply using the largest possible size: that of the bag itself. Using one of the other values sharpens the analysis, but does not change it in outline.

Sidebar: The Hypergeometric Distribution. The *hypergeometric distribution* is a discrete probability distribution constructed by sampling a universe of N items without replacement. It is usually explained as the distribution of probabilities among the number of successes in a sample of n items, where m of the N items in the universe count as successes. For our analysis, however, it is more intuitive to use an equivalent formulation based on the size of the overlap between two subsets of sizes n and m. In this formulation, the hypergeometric distribution $h_{<N, n, m>}$ is the distribution of probabilities over all possible sizes of the intersection, from 0 to the the lesser of n and m. We will use the notation $h_{<N, n, m>}(x)$, $0 \le x \le \min(n, m)$, to denote the probability of the two subsets sharing exactly x "hits." This probability is given by the equation

$$h_{<N, n, m>}(x) = \frac{\binom{m}{x}\binom{N-m}{n-x}}{\binom{N}{n}}$$

$$= \frac{n!\cdot m!\cdot(N-n)!\cdot(N-m)!}{x!\cdot(n-x)!\cdot(m-x)!\cdot N!\cdot(N-n-m+x)!}$$

We are often intereseted in the probability that two subsets are disjoint; that they have no overlap. This value is given by

$$h_{<N,\,n,\,m>}(0) = \frac{\binom{N-m}{n}}{\binom{N}{n}}$$

$$= \frac{(N-n)!\cdot(N-m)!}{N!\cdot(N-n-m)!}$$

Computationally, this formula looks like m factors (or n factors; we choose the smaller number for obvious reasons) of a steadily decreasing fraction:

$$= \frac{(N-n)\cdot(N-1)\cdot \ldots \cdot(N-n-m+1)}{N\cdot(n-1)\cdot \ldots \cdot(N-m+1)}$$

$$=\sim \quad ((N-n)/N)^m$$

(Where this last, of course, is the value for the binomial distribution obtained by taking two samples *with replacement* from a universe of N items.) The larger N is with respect to n and m, the better the binomial distribution approximates the hypergeometric.

Say we want to know whether the top 20 elements of an accumulator bank are stable at the .01 level; that is, whether there is a 99% chance that they will be unaffected by exploring the remainder of a wtdObjSet collection. Let us assume a universe of 1,000,000 objects and ask how small the unexplored tail needs to be to obtain this level of stability. Then we are want to find an m for which

$$h_{<1000000,\,20,\,m>}(0) \geq .99$$

Several terms in the factorial version of the equation drop out in the case where x=0, and we find that the largest m for which the inequality holds is 502.

[[**Check:** let us assume for the moment that the wtdObjSet collection sample really is drawn with replacement. Then the probability that each element misses the top 20 elements is:

(1,000,000-20)/1,000,000 = .99998

and the probability that the entire sample is disjoint is

(.99998)^m

The largest m for which this quantity is greater than .99 is 502.]]

Remember that we are dealing here with wtdObjSets of thousands, even tens of thousands of elements, and wtdObjSet collections of hundreds of thousands of elements. It is these numbers that drive us to use opportunistic methods. Once we have reduced such a collection to the point where only 500 elements remain, we are virtually done, and

might as well load in the last few. The amount of computation time that has been spent in calculating probabilities and maintaining the extra structure needed for stop-rule search is already more than would be required to run 500 elements into an accumulator bank. Thus we conclude that to satisfy this criterion of stability we are best off doing an exhaustive exploration of the component set collection.

In the context of approximate matching, we can justify relaxing this criterion. Consider the case of a simple text search, where a user is attempting to match a vector of terms across a text collection. No known ranking function does a very good job of ordering the results of such a search [RESEARCH?], although many correlate well over a large sample with the judgements of trained searchers. So we may justifiably believe that we do not need to establish the precise ranking that our function would predict *if we can be certain that the best documents have are in the top portion of the set.* In other words, if we have established the content of the top portion correctly, we can afford to be somewhat relaxed about the order.

It is one of the features of the above equation that even for large N only small sets are at all likely to be disjoint. Fig. PROB_DISJOINT, for instance, shows the probability that two samples drawn from a universe of 10,000 events will be disjoint. As you can see, unless at least one sample is very small (less than a few hundred elements) there is virtually no chance that the pair will be disjoint. This “handball court” shape is less pronounced with smaller total universes, but the effect is the same: unless the samples are extremely small, the chances are that they will have some overlap.

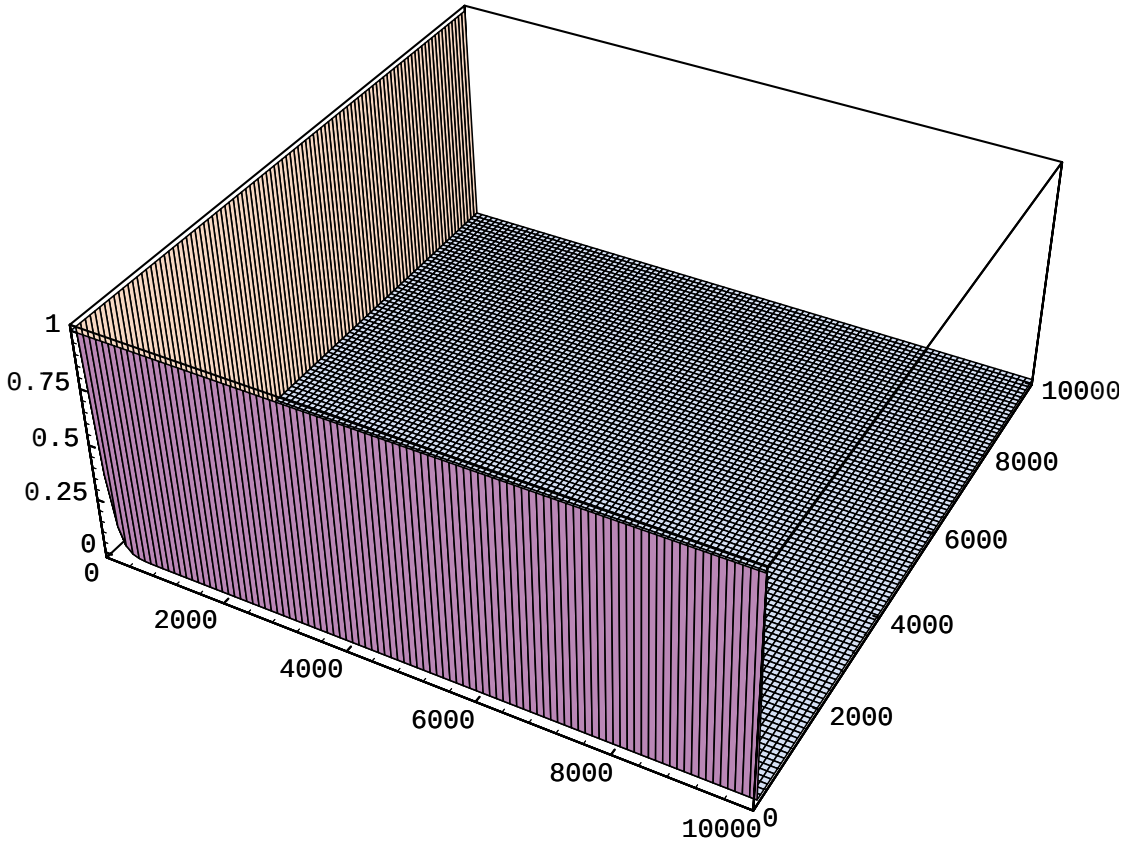

Fig. PROB_DISJOINT: The "handball court" effect for the probability of two lists being disjoint. Shown here are list sizes from 1 to N (10,000) on the two horizontal axes, and the probability that the two lists are disjoint on the vertical axis.

The second of our tests asks whether any elements of the unexplored portion of the component set collection are likely to belong in the top portion of the combined set. This question breaks down into two: is it possible to achieve a total weight greater than the last element in the top portion by summing the weights from the component sets; and what is the likelihood that such a summation should occur, i.e, that a single element might occur in all the lists? The first question is easy to answer. We simply sum the expected weights across (the remainder of) each component set. We will defer discussion of the second question to our discussion of the third test, where it occurs in a more [[highly constrained]] context. In fact, the second test is really just a special case of the third test, and any exploration that passes the third will pass the second.

The third test constrains the probability of a promotion from the lower portion of the merged set into the top portion. We want to know whether the dividing line between these two sections of the wtdObjSet is **stable** in the following technical sense:

Definition: A point in the merged portion of a summative searcher is stable if there are none of the items below that point are likely to be promoted above it during exploration of the remainder of the component set collection. In particular, a point is stable if the total probability of such a promoting hit is less than a given cutoff.

Note that a stable point is exactly that: a point. A point a few elements to one side or the other may not be stable, particularly if that point is on a riser in a stairstep attenuation function. This is a restatement of our result in the discussion of the first test above. Just because a section of the combined list is stable – unlikely to receive any new elements during the remainder of the exploration – does not mean that it is static – that elements within the section are not likely to be reordered by further exploration.

We break the third test into a set of tests. First, we consider single hits from the remainder of the component set collection, and ask which of these are likely to move an element from the tail of the combined set into the head. This is a sampling question, but not a simply a question of the probability of overlap between the unexplored collection and the tail. It may well be the case that most of the weights in both the unexplored portion and the combined tail are very low, much smaller than our candidate point of stability. We are concerned only with a subspace of all the possible hits and misses: with only those hits whose combined weight is greater than the candidate point.

In a state of complete information about both the component set collection and the merged set, we could make an exact count of all the promoting hits. Of course, in a state of exact information, we would not need an opportunistic search. We cannot, for instance, know the exact drop-off of the collection, so we do not know exactly how many elements in the collection have weight great enough to send, for instance, the top element of the combined tail into the head. We do have an expected collection attenuation function, however, and we can know both the maximum and the expected value for a single item in that collection.

The first calculation in our set, then, is of the probability that the expected weight of a collection item will promote any item in the combined tail. In particular, if there are **N** possible items in the collection as a whole, **c** items remaining in the component set collection and **d** items in the tail with weight greater than the difference between the expected weight and the weight of the last item in the top portion, we want to know if

$$h_{<N,\ c,\ d>}(0) > P_{cutoff}.$$

We then ask the same question for items that occur in several of the component sets. If an item occurs in two of the component sets, then we need to know if the sum of the expected weights for those sets is likely to cause a promotion. Since the total hit weight is higher, we will need to include a larger portion of the tail in our one sample; however, the expected number of items in common between component sets is likely to be much lower. In fact, the mean overlap between component sets only grows noticeable as product of the size of the two sets approaches N, and never grows large compared to N (Fig. MEAN_OVERLAP). The hypergeometric distribution has a greater spread than the binomial, however, so the mean may not be the best measure. A more important value might be the largest overlap within a certain probability.

Practically speaking, we generally only need to ask the question for pairs of component sets. To obtain a triple hit requires a hit by an element of the third component set on the intersection of the other two. Since that intersection is small, the chances of largest three-way overlap within our target probability interval can typically be counted on the fingers of one hand. The probabilities of four-way and greater intersections get correspondingly smaller. [[Calculate this to be sure, perhaps for the case where all the ni are equal.]]

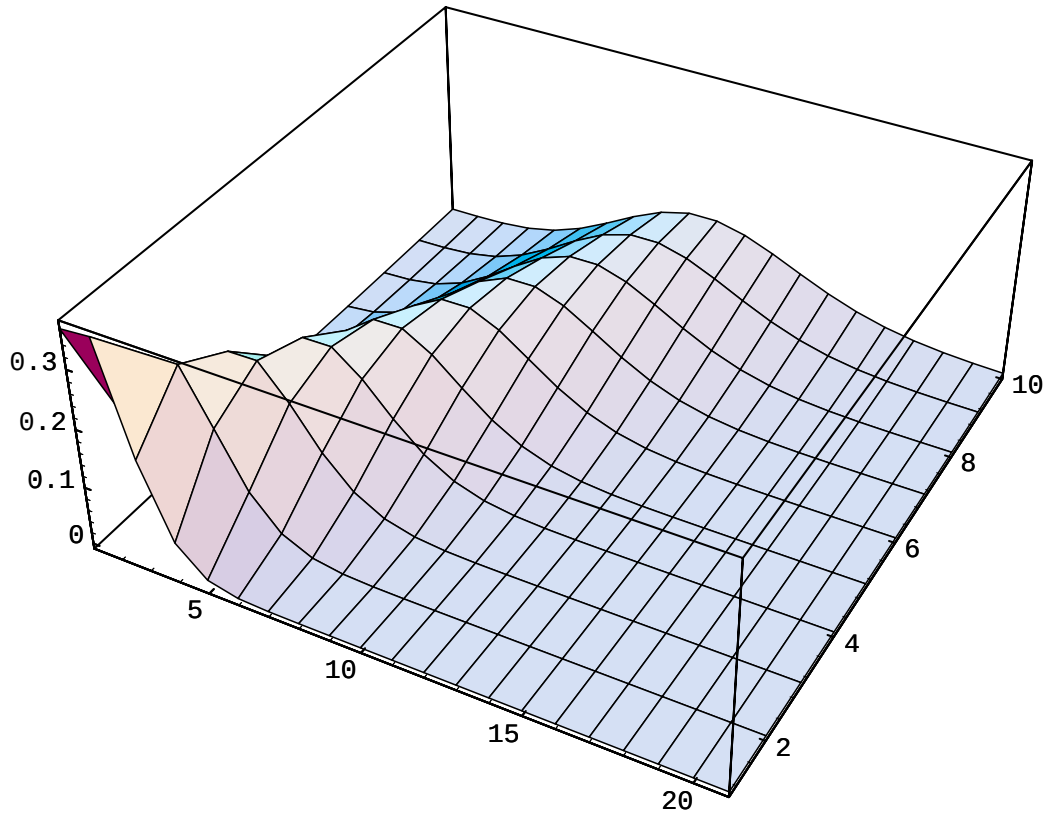

Fig. MEAN_OVERLAP: Probability density surface for overlaps between a 1000-element set and sets of sizes from 100 to 1000 (points 1 to 10 on the z axis) in a universe of 100,000 items. For the case of $h_{<100000,\ 1000,\ 100>}$, overlaps of 0 and 1 are most likely. By $h_{<100000,\ 1000,\ 1000>}$, the mean overlap is about 10 items. Even in this case, though, the probability of an overlap of size 20 is still less than .002.

[[**stable points and stable areas** An opportunistic searcher has a single list threading through the single-hit category, together with a pointer into the list at a point above which it is deemed to be stable by the statistical "stopping rules" that make the searcher opportunistic. In the Academy, this point of stability is defined to be the lowest point in the weight ordering at which no object from the lower segment could be promoted into the higher segment by additional hits from the component sets. In particular, in a universe of **n** possible objects, if there are **c** items remaining in the component sets, if the expected weight of any of those **c** items is **w**, and if there are **d** items of expected weight **x** below a particular point in the main list, then that point is a point of stability if:

$$(x+w)*p < s$$

where **s** is the weight of the point itself, and

$$p = 1 - ((n-c)! * (n-d)! \ / n! * (n-c-d)!)$$

is the probability that a hit will occur in the lower segment.

ALGORITHM: First we make a pair of quick and cheap tests. If SingleHit is empty, then we have to be able to stop, because there's nothing more to search. Alternatively, if Point is at the very bottom of the accumulator, then we have no way to tell whether we can stop yet, so we have to explore farther.

Then come the two hard and expensive parts. We need to know: is it likely (within probCutOff) that we will find a previously unseen object in SingleHit that belongs among the top Depth objects in A? And we need to know if it's likely (within probCutOff) that an object or objects still in SingleHit will hit a member of A below the top Depth with enough additional weight to send it into the top portion.

Both of these take some calculation: the latter is done here and the former in wosetColl::expMaxCommonDocs(), but they use similar equations (see documentation in prob_disjoint.cc). If either test fails, we return FALSE; if both succeed, we can return TRUE.

It is not clear which test should be done first. At the present, I'm doing the former first, both because I think it may be cheaper more often, and because I think it is the more likely to fail. But I could be wrong about either, in which case the other order might work better.

]]

The preceding analysis presumes statistical independence of the component wtdObjSets. This is a common assumption – and in fact the only assumption that allows a tractable analysis – but in point of fact it is completely mistaken. Words in English have complex networks of interdependency [VectModel; Church]. Words in a query are if anything less independent. We might say that users construct queries in the fervent hope that their search terms are *not* independent; that they are likely to co-occur in documents of interest.

Suppose we are searching a moderate-size research library collection (the Virginia Tech collection of 1994, with a total of 900,000 volumes) using the query **Author: Asimov; Title: Foundation**. In the Virginia Tech collection there are two authors named Asimov: Isaac, with a total of 137 books to his credit, and Janet with two. The word "foundation" in the title retrieves 3752 [[CHECK!]] books, including nine by Asimov. Let us pretend that these two keys are statistically independent and calculate the odds of any overlap at all.

As developed above, the odds of an overlap of x works are

$$h_{<900000,\ 139,\ 3572>}(x) \qquad 0 \le x \le 139$$

With these parameters, the odds are 0.56 that the two sets have no intersection at all. Above x=3 the odds become miniscule, and the probability of an overlap of nine works calculates to 9.00×10^{-9}. Such an event thus satisfies the most stringent tests for statistical significance – not surprisingly, since there is in fact a [[significant]] correlation [[in the real world]] between Isaac Asimov and "Foundation." It is that correlation that informed the query, and that undermines any statistical rules for curtailing exploration.

The Occam searcher attempts to sidestep this problem by beginning its search with an extremely liberal sample, combining all component objects with scaled weights down to 0.5 on the scale [0,1]. This has the dual effect of removing the top segment of very large

component sets and completely eliminating smaller sets. When the searcher is combining terms from a single text collection weighted with a strict IDF function, for instance, the initial run would include all terms associated with text sets the square root of the collection size in cardinality. In a million document collection, this would include all terms that occurs in less than 1000 documents. [[other weighting schemes?]]

This system works well in practice. We hypothesize that there is some relationship between frequency and word associations: that higher frequency words are less likely to be semantically bound to specific other words than their rarer cousins. This is itself only a tendency statement. Even high frequency prepositions like *of* and *on* show marked preferences for some words over others [Church], though not as high as *Asimov* with *foundation.* [[Moreover,]] Over a sample of a sentence or a paragraph [[, though,]] such preferences tend to attenuate. It would be most interesting to run a statistical study comparing strong associations over such spans with the frequency of the associated words.