Scaling Distributional Similarity to Large Corpora

James Gorman and **James R. Curran**
School of Information Technologies
University of Sydney
NSW 2006, Australia
{jgorman2,james}@it.usyd.edu.au

Abstract

Accurately representing synonymy using distributional similarity requires large volumes of data to reliably represent infrequent words. However, the naïve nearest-neighbour approach to comparing context vectors extracted from large corpora scales poorly ($O(n^2)$ in the vocabulary size).

In this paper, we compare several existing approaches to approximating the nearest-neighbour search for distributional similarity. We investigate the trade-off between efficiency and accuracy, and find that SASH (Houle and Sakuma, 2005) provides the best balance.

1 Introduction

It is a general property of Machine Learning that increasing the volume of training data increases the accuracy of results. This is no more evident than in Natural Language Processing (NLP), where massive quantities of text are required to model rare language events. Despite the rapid increase in computational power available for NLP systems, the volume of raw data available still outweighs our ability to process it. *Unsupervised learning*, which does not require the expensive and time-consuming human annotation of data, offers an opportunity to use this wealth of data. Curran and Moens (2002) show that synonymy extraction for lexical semantic resources using *distributional similarity* produces continuing gains in accuracy as the volume of input data increases.

Extracting synonymy relations using distributional similarity is based on the *distributional hypothesis* that *similar words appear in similar contexts*. Terms are described by collating information about their occurrence in a corpus into vectors. These *context vectors* are then compared for similarity. Existing approaches differ primarily in their definition of "context", e.g. the surrounding words or the entire document, and their choice of distance metric for calculating similarity between the context vectors representing each term.

Manual creation of lexical semantic resources is open to the problems of bias, inconsistency and limited coverage. It is difficult to account for the needs of the many domains in which NLP techniques are now being applied and for the rapid change in language use. The assisted or automatic creation and maintenance of these resources would be of great advantage.

Finding synonyms using distributional similarity requires a nearest-neighbour search over the context vectors of each term. This is computationally intensive, scaling to $O(n^2m)$ for the number of terms n and the size of their context vectors m. Increasing the volume of input data will increase the size of both n and m, decreasing the efficiency of a naïve nearest-neighbour approach.

Many approaches to reduce this complexity have been suggested. In this paper we evaluate state-of-the-art techniques proposed to solve this problem. We find that the Spatial Approximation Sample Hierarchy (Houle and Sakuma, 2005) provides the best accuracy/efficiency trade-off.

2 Distributional Similarity

Measuring distributional similarity first requires the extraction of context information for each of the vocabulary terms from raw text. These terms are then compared for similarity using a nearest-neighbour search or clustering based on distance calculations between the statistical descriptions of their contexts.

2.1 Extraction

A *context relation* is defined as a tuple (w, r, w') where w is a term, which occurs in some grammatical relation r with another word w' in some sentence. We refer to the tuple (r, w') as an *attribute* of w. For example, (dog, direct-obj, walk) indicates that dog was the direct object of walk in a sentence.

In our experiments context extraction begins with a Maximum Entropy POS tagger and chunker. The SEXTANT relation extractor (Grefenstette, 1994) produces context relations that are then lemmatised. The relations for each term are collected together and counted, producing a vector of attributes and their frequencies in the corpus.

2.2 Measures and Weights

Both nearest-neighbour and cluster analysis methods require a distance measure to calculate the similarity between context vectors. Curran (2004) decomposes this into *measure* and *weight* functions. The *measure* calculates the similarity between two weighted context vectors and the *weight* calculates the informativeness of each context relation from the raw frequencies.

For these experiments we use the Jaccard (1) measure and the TTest (2) weight functions, found by Curran (2004) to have the best performance.

$$\frac{\sum_{(r,w')} \min(\mathrm{w}(w_m, r, w'), \mathrm{w}(w_n, r, w'))}{\sum_{(r,w')} \max(\mathrm{w}(w_m, r, w'), \mathrm{w}(w_n, r, w'))} \tag{1}$$

$$\frac{p(w, r, w') - p(*, r, w')p(w, *, *)}{\sqrt{p(*, r, w')p(w, *, *)}} \tag{2}$$

2.3 Nearest-neighbour Search

The simplest algorithm for finding synonyms is a k-nearest-neighbour (k-NN) search, which involves pair-wise vector comparison of the target term with every term in the vocabulary. Given an n term vocabulary and up to m attributes for each term, the asymptotic time complexity of nearest-neighbour search is $O(n^2m)$. This is very expensive, with even a moderate vocabulary making the use of huge datasets infeasible. Our largest experiments used a vocabulary of over 184,000 words.

3 Dimensionality Reduction

Using a cut-off to remove low frequency terms can significantly reduce the value of n. Unfortunately, reducing m by eliminating low frequency contexts has a significant impact on the quality of the results. There are many techniques to reduce dimensionality while avoiding this problem. The simplest methods use feature selection techniques, such as information gain, to remove the attributes that are less informative. Other techniques smooth the data while reducing dimensionality.

Latent Semantic Analysis (LSA, Landauer and Dumais, 1997) is a smoothing and dimensionality reduction technique based on the intuition that the true dimensionality of data is *latent* in the surface dimensionality. Landauer and Dumais admit that, from a pragmatic perspective, the same effect as LSA can be generated by using large volumes of data with very long attribute vectors. Experiments with LSA typically use attribute vectors of a dimensionality of around 1000. Our experiments have a dimensionality of 500,000 to 1,500,000. Decompositions on data this size are computationally difficult. Dimensionality reduction is often used before using LSA to improve its scalability.

3.1 Heuristics

Another technique is to use an initial heuristic comparison to reduce the number of full $O(m)$ vector comparisons that are performed. If the heuristic comparison is sufficiently fast and a sufficient number of full comparisons are avoided, the cost of an additional check will be easily absorbed by the savings made.

Curran and Moens (2002) introduces a vector of *canonical* attributes (of bounded length $k \ll m$), selected from the full vector, to represent the term. These attributes are the most strongly weighted verb attributes, chosen because they constrain the semantics of the term more and partake in fewer idiomatic collocations. If a pair of terms share at least one canonical attribute then a full similarity comparison is performed, otherwise the terms are not compared. They show an 89% reduction in search time, with only a 3.9% loss in accuracy.

There is a significant improvement in the computational complexity. If a maximum of p positive results are returned, our complexity becomes $O(n^2k + npm)$. When $p \ll n$, the system will be faster as many fewer full comparisons will be made, but at the cost of accuracy as more possibly near results will be discarded out of hand.

4 Randomised Techniques

Conventional dimensionality reduction techniques can be computationally expensive: a more scalable solution is required to handle the volumes of data we propose to use. Randomised techniques provide a possible solution to this.

We present two techniques that have been used recently for distributional similarity: Random Indexing (Kanerva et al., 2000) and Locality Sensitive Hashing (LSH, Broder, 1997).

4.1 Random Indexing

Random Indexing (RI) is a hashing technique based on *Sparse Distributed Memory* (Kanerva, 1993). Karlgren and Sahlgren (2001) showed RI produces results similar to LSA using the *Test of English as a Foreign Language* (TOEFL) evaluation. Sahlgren and Karlgren (2005) showed the technique to be successful in generating bilingual lexicons from parallel corpora.

In RI, we first allocate a d length *index vector* to each unique attribute. The vectors consist of a large number of 0s and small number (ϵ) number of randomly distributed ± 1s. *Context vectors*, identifying terms, are generated by summing the index vectors of the attributes for each non-unique context in which a term appears. The context vector for a term t appearing in contexts $c_1 = [1, 0, 0, -1]$ and $c_2 = [0, 1, 0, -1]$ would be $[1, 1, 0, -2]$. The distance between these context vectors is then measured using the cosine measure:

$$\cos(\theta(u, v)) = \frac{\vec{u} \cdot \vec{v}}{|\vec{u}|\ |\vec{v}|} \tag{3}$$

This technique allows for incremental sampling, where the index vector for an attribute is only generated when the attribute is encountered. Construction complexity is $O(nmd)$ and search complexity is $O(n^2 d)$.

4.2 Locality Sensitive Hashing

LSH is a probabilistic technique that allows the approximation of a similarity function. Broder (1997) proposed an approximation of the Jaccard similarity function using min-wise independent functions. Charikar (2002) proposed an approximation of the cosine measure using random hyperplanes Ravichandran et al. (2005) used this cosine variant and showed it to produce over 70% accuracy in extracting synonyms when compared against Pantel and Lin (2002).

Given we have n terms in an m' dimensional space, we create $d \ll m'$ unit random vectors also of m' dimensions, labelled $\{\vec{r_1}, \vec{r_2}, ..., \vec{r_d}\}$. Each vector is created by sampling a Gaussian function m' times, with a mean of 0 and a variance of 1.

For each term w we construct its bit signature using the function

$$h_{\vec{r}}(\vec{w}) = \begin{cases} 1 & : \vec{r}.\vec{w} \geq 0 \\ 0 & : \vec{r}.\vec{w} < 0 \end{cases}$$

where $\vec{r}$ is a spherically symmetric random vector of length d. The signature, $\bar{w}$, is the d length bit vector:

$$\bar{w} = \{h_{\vec{r_1}}(\vec{w}), h_{\vec{r_2}}(\vec{w}), \ldots, h_{\vec{r_d}}(\vec{w})\}$$

The cost to build all n signatures is $O(nm'd)$.

For terms u and v, Goemans and Williamson (1995) approximate the angular similarity by

$$p(h_{\vec{r}}(\vec{u}) = h_{\vec{r}}(\vec{v})) = 1 - \frac{\theta(\vec{u}, \vec{u})}{\pi} \tag{4}$$

where $\theta(\vec{u}, \vec{u})$ is the angle between $\vec{u}$ and $\vec{u}$. The angular similarity gives the cosine by

$$\begin{aligned} &cos(\theta(\vec{u}, \vec{u})) = \\ &\quad cos((1 - p(h_{\vec{r}}(\vec{u}) = h_{\vec{r}}(\vec{v})))\pi) \end{aligned} \tag{5}$$

The probability can be derived from the Hamming distance:

$$p(h_r(u) = h_r(v)) = 1 - \frac{\mathcal{H}(\bar{u}, \bar{v})}{d} \tag{6}$$

By combining equations 5 and 6 we get the following approximation of the cosine distance:

$$cos(\theta(\vec{u}, \vec{u})) = cos\left(\left(\frac{\mathcal{H}(\bar{u}, \bar{v})}{d}\right)\pi\right) \tag{7}$$

That is, the cosine of two context vectors is approximated by the cosine of the Hamming distance between their two signatures normalised by the size of the signatures. Search is performed using Equation 7 and scales to $O(n^2 d)$.

5 Data Structures

The methods presented above fail to address the n^2 component of the search complexity. Many data structures have been proposed that can be used to address this problem in similarity searching. We present three data structures: the *vantage point tree* (VPT, Yianilos, 1993), which indexes points in a metric space, *Point Location in Equal*

Balls (PLEB, Indyk and Motwani, 1998), a probabilistic structure that uses the bit signatures generated by LSH, and the *Spatial Approximation Sample Hierarchy* (SASH, Houle and Sakuma, 2005), which approximates a k-NN search.

Another option inspired by IR is attribute indexing (INDEX). In this technique, in addition to each term having a reference to its attributes, each attribute has a reference to the terms referencing it. Each term is then only compared with the terms with which it shares attributes. We will give a theoretically comparison against other techniques.

5.1 Vantage Point Tree

Metric space data structures provide a solution to near-neighbour searches in very high dimensions. These rely solely on the existence of a comparison function that satisfies the conditions of metricality: non-negativity, equality, symmetry and the triangle inequality.

VPT is typical of these structures and has been used successfully in many applications. The VPT is a binary tree designed for range searches. These are searches limited to some distance from the target term but can be modified for k-NN search.

VPT is constructed recursively. Beginning with a set of U terms, we take any term to be our vantage point p. This becomes our root. We now find the median distance m_p of all other terms to p: $m_p = median\{dist(p,u)|u \in U\}$. Those terms u such that $dist(p,u) \leq m_p$ are inserted into the left sub-tree, and the remainder into the right subtree. Each sub-tree is then constructed as a new VPT, choosing a new vantage point from within its terms, until all terms are exhausted.

Searching a VPT is also recursive. Given a term q and radius r, we begin by measuring the distance to the root term p. If $dist(q,p) \leq r$ we enter p into our list of near terms. If $dist(q,p) - r \leq m_p$ we enter the left sub-tree and if $dist(q,p) + r > mp$ we enter the right sub-tree. Both sub-trees may be entered. The process is repeated for each entered subtree, taking the vantage point of the sub-tree to be the new root term.

To perform a k-NN search we use a *backtracking decreasing radius search* (Burkhard and Keller, 1973). The search begins with $r = \infty$, and terms are added to a list of the closest k terms. When the k^{th} closest term is found, the radius is set to the distance between this term and the target. Each time a new, closer element is added to the list, the radius is updated to the distance from the target to the new k^{th} closest term.

Construction complexity is $O(n \log n)$. Search complexity is claimed to be $O(\log n)$ for small radius searches. This does not hold for our decreasing radius search, whose worst case complexity is $O(n)$.

5.2 Point Location in Equal Balls

PLEB is a randomised structure that uses the bit signatures generated by LSH. It was used by Ravichandran et al. (2005) to improve the efficiency of distributional similarity calculations.

Having generated our d length bit signatures for each of our n terms, we take these signatures and randomly permute the bits. Each vector has the same permutation applied. This is equivalent to a column reordering in a matrix where the rows are the terms and the columns the bits. After applying the permutation, the list of terms is sorted lexicographically based on the bit signatures. The list is scanned sequentially, and each term is compared to its B nearest neighbours in the list. The choice of B will effect the accuracy/efficiency trade-off, and need not be related to the choice of k. This is performed q times, using a different random permutation function each time. After each iteration, the current closest k terms are stored.

For a fixed d, the complexity for the permutation step is $O(qn)$, the sorting $O(qn \log n)$ and the search $O(qBn)$.

5.3 Spatial Approximation Sample Hierarchy

SASH approximates a k-NN search by precomputing some near neighbours for each node (terms in our case). This produces multiple paths between terms, allowing SASH to shape itself to the data set (Houle, 2003). The following description is adapted from Houle and Sakuma (2005).

The SASH is a directed, edge-weighted graph with the following properties (see Figure 1):

- Each term corresponds to a unique node.
- The nodes are arranged into a hierarchy of levels, with the bottom level containing $\frac{n}{2}$ nodes and the top containing a single root node. Each level, except the top, will contain half as many nodes as the level below.
- Edges between nodes are linked to consecutive levels. Each node will have at most p *parent* nodes in the level above, and c *child* nodes in the level below.

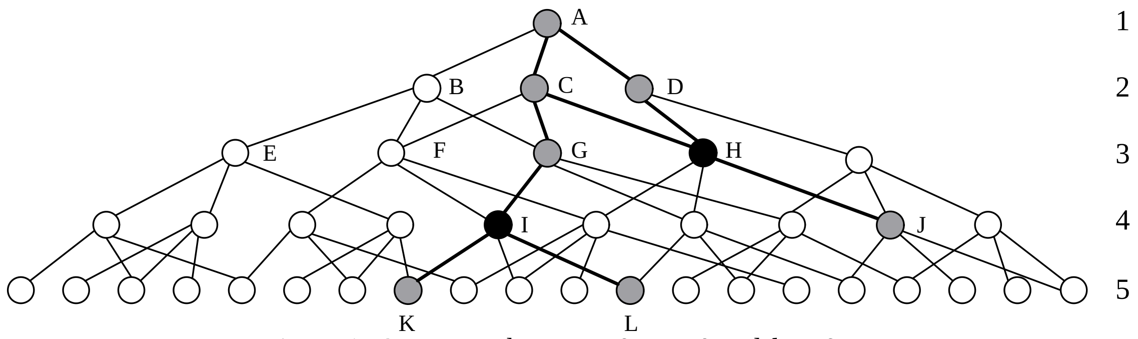

Figure 1: A SASH, where $p = 2$, $c = 3$ and $k = 2$

- Every node must have at least one parent so that all nodes are reachable from the root.

Construction begins with the nodes being randomly distributed between the levels. SASH is then constructed iteratively by each node finding its closest p parents in the level above. The parent will keep the closest c of these children, forming edges in the graph, and reject the rest. Any nodes without parents after being rejected are then assigned as children of the nearest node in the previous level with fewer than c children.

Searching is performed by finding the k nearest nodes at each level, which are added to a set of near nodes. To limit the search, only those nodes whose parents were found to be nearest at the previous level are searched. The k closest nodes from the set of near nodes are then returned. The search complexity is $O(ck \log n)$.

In Figure 1, the filled nodes demonstrate a search for the near-neighbours of some node q, using $k = 2$. Our search begins with the root node A. As we are using $k = 2$, we must find the two nearest children of A using our similarity measure. In this case, C and D are closer than B. We now find the closest two children of C and D. E is not checked as it is only a child of B. All other nodes are checked, including F and G, which are shared as children by B and C. From this level we chose G and H. The final levels are considered similarly.

At this point we now have the list of near nodes A, C, D, G, H, I, J, K and L. From this we chose the two nodes nearest q, H and I marked in black, which are then returned.

k can be varied at each level to force a larger number of elements to be tested at the base of the SASH using, for instance, the equation:

$$k_i = \max\{ k^{1-\frac{h-i}{\log n}}, \frac{1}{2}pc \} \tag{8}$$

This changes our search complexity to:

$$\frac{k^{1+\frac{1}{\log n}}}{k^{\frac{1}{\log n}}-1} + \frac{pc^2}{2}\log n \tag{9}$$

We use this geometric function in our experiments.

Gorman and Curran (2005a; 2005b) found the performance of SASH for distributional similarity could be improved by replacing the initial random ordering with a frequency based ordering. In accordance with Zipf's law, the majority of terms have low frequencies. Comparisons made with these low frequency terms are unreliable (Curran and Moens, 2002). Creating SASH with high frequency terms near the root produces more reliable initial paths, but comparisons against these terms are more expensive.

The best accuracy/efficiency trade-off was found when using *more* reliable initial paths rather than the *most* reliable. This is done by *folding* the data around some mean number of relations. For each term, if its number of relations m_i is greater than some chosen number of relations $\mathcal{M}$, it is given a new ranking based on the score $\frac{\mathcal{M}^2}{m_i}$. Otherwise its ranking based on its number of relations. This has the effect of pushing very high and very low frequency terms away from the root.

6 Evaluation Measures

The simplest method for evaluation is the direct comparison of extracted synonyms with a manually created gold standard (Grefenstette, 1994). To reduce the problem of limited coverage, our evaluation combines three electronic thesauri: the Macquarie, Roget's and Moby thesauri.

We follow Curran (2004) and use two performance measures: direct matches (DIRECT) and inverse rank (INVR). DIRECT is the percentage of returned synonyms found in the gold standard. INVR is the sum of the inverse rank of each matching synonym, e.g. matches at ranks 3, 5 and 28

CORPUS	CUT-OFF	TERMS	AVERAGE RELATIONS PER TERM
BNC	0	246,067	43
	5	88,926	116
	100	14,862	617
LARGE	0	541,722	97
	5	184,494	281
	100	35,618	1,400

Table 1: Extracted Context Information

give an inverse rank score of $\frac{1}{3} + \frac{1}{5} + \frac{1}{28}$. With at most 100 matching synonyms, the maximum INVR is 5.187. This more fine grained as it incorporates the both the number of matches and their ranking. The same 300 single word nouns were used for evaluation as used by Curran (2004) for his large scale evaluation. These were chosen randomly from WordNet such that they covered a range over the following properties: *frequency*, *number of senses*, *specificity* and *concreteness*. For each of these terms, the closest 100 terms and their similarity scores were extracted.

7 Experiments

We use two corpora in our experiments: the smaller is the non-speech portion of the British National Corpus (BNC), 90 million words covering a wide range of domains and formats; the larger consists of the BNC, the Reuters Corpus Volume 1 and most of the English news holdings of the LDC in 2003, representing over 2 billion words of text (LARGE, Curran, 2004).

The semantic similarity system implemented by Curran (2004) provides our baseline. This performs a brute-force k-NN search (NAIVE). We present results for the canonical attribute heuristic (HEURISTIC), RI, LSH, PLEB, VPT and SASH.

We take the optimal canonical attribute vector length of 30 for HEURISTIC from Curran (2004). For SASH we take optimal values of $p = 4$ and $c = 16$ and use the folded ordering taking $\mathcal{M} = 1000$ from Gorman and Curran (2005b).

For RI, LSH and PLEB we found optimal values experimentally using the BNC. For LSH we chose $d = 3,000$ (LSH$_{3,000}$) and $10,000$ (LSH$_{10,000}$), showing the effect of changing the dimensionality. The frequency statistics were weighted using mutual information, as in Ravichandran et al. (2005):

$$\log(\frac{p(w,r,w')}{p(w,*,*)p(*,r,w')}) \qquad (10)$$

PLEB used the values $q = 500$ and $B = 100$.

	CUT-OFF 5	CUT-OFF 100
NAIVE	**1.72**	**1.71**
HEURISTIC	1.65	1.66
RI	0.80	0.93
LSH$_{10,000}$	1.26	1.31
SASH	**1.73**	**1.71**

Table 2: INVR vs frequency cut-off

The initial experiments on RI produced quite poor results. The intuition was that this was caused by the lack of smoothing in the algorithm. Experiments were performed using the weights given in Curran (2004). Of these, mutual information (10), evaluated with an extra $\log_2(f(w,r,w') + 1)$ factor and limited to positive values, produced the best results (RI$_{\text{MI}}$). The values $d = 1000$ and $\epsilon = 5$ were found to produce the best results.

All experiments were performed on 3.2GHz Xeon P4 machines with 4GB of RAM.

8 Results

As the accuracy of comparisons between terms increases with frequency (Curran, 2004), applying a frequency cut-off will both reduce the size of the vocabulary (n) and increase the average accuracy of comparisons. Table 1 shows the reduction in vocabulary and increase in average context relations per term as cut-off increases. For LARGE, the initial 541,722 word vocabulary is reduced by 66% when a cut-off of 5 is applied and by 86% when the cut-off is increased to 100. The average number of relations increases from 97 to 1400.

The work by Curran (2004) largely uses a frequency cut-off of 5. When this cut-off was used with the randomised techniques RI and LSH, it produced quite poor results. When the cut-off was increased to 100, as used by Ravichandran et al. (2005), the results improved significantly. Table 2 shows the INVR scores for our various techniques using the BNC with cut-offs of 5 and 100.

Table 3 shows the results of a full thesaurus extraction using the BNC and LARGE corpora using a cut-off of 100. The average DIRECT score and INVR are from the 300 test words. The total execution time is extrapolated from the average search time of these test words and includes the setup time. For LARGE, extraction using NAIVE takes 444 hours: over 18 days. If the 184,494 word vocabulary were used, it would take over 7000 hours, or nearly 300 days. This gives some indication of

	BNC			LARGE		
	DIRECT	INVR	Time	DIRECT	INVR	Time
NAIVE	5.23	1.71	38.0hr	5.70	1.93	444.3hr
HEURISTIC	4.94	1.66	2.0hr	5.51	1.93	30.2hr
RI	2.97	0.93	0.4hr	2.42	0.85	1.9hr
RI_{MI}	**3.49**	**1.41**	**0.4hr**	**4.58**	**1.75**	**1.9hr**
$\text{LSH}_{3,000}$	2.00	0.76	0.7hr	2.92	1.07	3.6hr
$\text{LSH}_{10,000}$	3.68	1.31	2.3hr	3.77	1.40	8.4hr
$\text{PLEB}_{3,000}$	2.00	0.76	1.2hr	2.85	1.07	4.1hr
$\text{PLEB}_{10,000}$	3.66	1.30	3.9hr	3.63	1.37	11.8hr
VPT	5.23	1.71	15.9hr	5.70	1.93	336.1hr
SASH	**5.17**	**1.71**	**2.0hr**	**5.29**	**1.89**	**23.7hr**

Table 3: Full thesaurus extraction

the scale of the problem.

The only technique to become less accurate when the corpus size is increased is RI; it is likely that RI is sensitive to high frequency, low information contexts that are more prevalent in LARGE. Weighting reduces this effect, improving accuracy.

The importance of the choice of d can be seen in the results for LSH. While much slower, $\text{LSH}_{10,000}$ is also much more accurate than $\text{LSH}_{3,000}$, while still being much faster than NAIVE. Introducing the PLEB data structure does not improve the efficiency while incurring a small cost on accuracy. We are not using large enough datasets to show the improved time complexity using PLEB.

VPT is only slightly faster slightly faster than NAIVE. This is not surprising in light of the original design of the data structure: decreasing radius search does not guarantee search efficiency.

A significant influence in the speed of the randomised techniques, RI and LSH, is the fixed dimensionality. The randomised techniques use a fixed length vector which is not influenced by the size of m. The drawback of this is that the size of the vector needs to be tuned to the dataset.

It would seem at first glance that HEURISTIC and SASH provide very similar results, with HEURISTIC slightly slower, but more accurate. This misses the difference in time complexity between the methods: HEURISTIC is n^2 and SASH $n \log n$. The improvement in execution time over NAIVE decreases as corpus size increases and this would be expected to continue. Further tuning of SASH parameters may improve its accuracy.

RI_{MI} produces similar result using LARGE to SASH using BNC. This does not include the cost of extracting context relations from the raw text, so the true comparison is much worse. SASH allows the free use of weight and measure functions, but RI is constrained by having to transform any context space into a RI space. This is important when considering that different tasks may require different weights and measures (Weeds and Weir, 2005). RI also suffers n^2 complexity, where as SASH is $n \log n$. Taking these into account, and that the improvements are barely significant, SASH is a better choice.

	LARGE		
CUT-OFF	0	5	100
NAIVE	541,721	184,493	35,617
SASH	10,599	8,796	6,231
INDEX	5,844	13,187	32,663

Table 4: Average number of comparisons per term

The results for LSH are disappointing. It performs consistently worse than the other methods except VPT. This could be improved by using larger bit vectors, but there is a limit to the size of these as they represent a significant memory overhead, particularly as the vocabulary increases.

Table 4 presents the theoretical analysis of attribute indexing. The average number of comparisons made for various cut-offs of LARGE are shown. NAIVE and INDEX are the actual values for those techniques. The values for SASH are *worst case*, where the maximum number of terms are compared at each level. The actual number of comparisons made will be much less. The efficiency of INDEX is sensitive to the density of attributes and increasing the cut-off increases the density. This is seen in the dramatic drop in performance as the cut-off increases. This problem of density will increase as volume of raw input data increases, further reducing its effectiveness. SASH is only dependent on the number of terms, not the density.

Where the need for computationally efficiency out-weighs the need for accuracy, RI_{MI} provides better results. SASH is the most balanced of the techniques tested and provides the most scalable, high quality results.

9 Conclusion

We have evaluated several state-of-the-art techniques for improving the efficiency of distributional similarity measurements. We found that, in terms of raw efficiency, Random Indexing (RI) was significantly faster than any other technique, but at the cost of accuracy. Even after our modifications to the RI algorithm to significantly improve its accuracy, SASH still provides a better accuracy/efficiency trade-off. This is more evident when considering the time to extract context information from the raw text. SASH, unlike RI, also allows us to choose both the weight and the measure used. LSH and PLEB could not match either the efficiency of RI or the accuracy of SASH.

We intend to use this knowledge to process even larger corpora to produce more accurate results.

Having set out to improve the efficiency of distributional similarity searches while limiting any loss in accuracy, we are producing full nearest-neighbour searches 18 times faster, with only a 2% loss in accuracy.

Acknowledgements

We would like to thank our reviewers for their helpful feedback and corrections. This work has been supported by the Australian Research Council under Discovery Project DP0453131.

References

Andrei Broder. 1997. On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences*, pages 21–29, Salerno, Italy.

Walter A. Burkhard and Robert M. Keller. 1973. Some approaches to best-match fi le searching. *Communications of the ACM*, 16(4):230–236, April.

Moses S. Charikar. 2002. Similarity estimation techniques from rounding algorithms. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 380–388, Montreal, Quebec, Canada, 19–21 May.

James Curran and Marc Moens. 2002. Improvements in automatic thesaurus extraction. In *Proceedings of the Workshop of the ACL Special Interest Group on the Lexicon*, pages 59–66, Philadelphia, PA, USA, 12 July.

James Curran. 2004. *From Distributional to Semantic Similarity*. Ph.D. thesis, University of Edinburgh.

Michel X. Goemans and David P. Williamson. 1995. Improved approximation algorithms for maximum cut and satisfi ability problems using semidefi nite programming. *Journal of Association for Computing Machinery*, 42(6):1115–1145, November.

James Gorman and James Curran. 2005a. Approximate searching for distributional similarity. In *ACL-SIGLEX 2005 Workshop on Deep Lexical Acquisition*, Ann Arbor, MI, USA, 30 June.

James Gorman and James Curran. 2005b. Augmenting approximate similarity searching with lexical information. In *Australasian Language Technology Workshop*, Sydney, Australia, 9–11 November.

Gregory Grefenstette. 1994. *Explorations in Automatic Thesaurus Discovery*. Kluwer Academic Publishers, Boston.

Michael E. Houle and Jun Sakuma. 2005. Fast approximate similarity search in extremely high-dimensional data sets. In *Proceedings of the 21st International Conference on Data Engineering*, pages 619–630, Tokyo, Japan.

Michael E. Houle. 2003. Navigating massive data sets via local clustering. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 547–552, Washington, DC, USA.

Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the 30th annual ACM Symposium on Theory of Computing*, pages 604–613, New York, NY, USA, 24–26 May. ACM Press.

Pentti Kanerva, Jan Kristoferson, and Anders Holst. 2000. Random indexing of text samples for latent semantic analysis. In *Proceedings of the 22nd Annual Conference of the Cognitive Science Society*, page 1036, Mahwah, NJ, USA.

Pentti Kanerva. 1993. Sparse distributed memory and related models. In M.H. Hassoun, editor, *Associative Neural Memories: Theory and Implementation*, pages 50–76. Oxford University Press, New York, NY, USA.

Jussi Karlgren and Magnus Sahlgren. 2001. From words to understanding. In Y. Uesaka, P. Kanerva, and H Asoh, editors, *Foundations of Real-World Intelligence*, pages 294–308. CSLI Publications, Stanford, CA, USA.

Thomas K. Landauer and Susan T. Dumais. 1997. A solution to plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological Review*, 104(2):211–240, April.

Patrick Pantel and Dekang Lin. 2002. Discovering word senses from text. In *Proceedings of ACM SIGKDD-02*, pages 613–619, 23–26 July.

Deepak Ravichandran, Patrick Pantel, and Eduard Hovy. 2005. Randomized algorithms and NLP: Using locality sensitive hash functions for high speed noun clustering. In *Proceedings of the 43rd Annual Meeting of the ACL*, pages 622–629, Ann Arbor, USA.

Mangus Sahlgren and Jussi Karlgren. 2005. Automatic bilingual lexicon acquisition using random indexing of parallel corpora. *Journal of Natural Language Engineering, Special Issue on Parallel Texts*, 11(3), June.

Julie Weeds and David Weir. 2005. Co-occurance retrieval: A fexible framework for lexical distributional similarity. *Computational Linguistics*, 31(4):439–475, December.

Peter N. Yianilos. 1993. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, pages 311–321, Philadelphia.