Appendix A. Properties of SimHash

Let a and b be vectors on the unit hypersphere in the Euclidean space $\mathcal{R}^d$ with angle θ between them, and let u be a random d-dimensional vector whose coordinates are sampled independently from $N(0,1)$. Note that the probability that the two vectors end up on a different side of the hyperplane is

$$\mathbb{P}(sign(a \cdot u) \neq sign(b \cdot u)) = 1 - \mathbb{P}(sign(a \cdot u) = sign(b \cdot u)) \tag{10}$$

$$= 1 - \frac{\theta}{\pi} \tag{11}$$

Let the random variable X have value 1 if a and b are on different sides of u, and 0 otherwise. Then:

$$\mathbb{E}[X] = 1 - \frac{\theta}{\pi} \tag{12}$$

$$\mathbb{V}[X] = \frac{\theta}{\pi}\left(1 - \frac{\theta}{\pi}\right) \tag{13}$$

If we let the family of random variables $X_1, ..., X_n$ be the result of repeating this process several times with independently randomly chosen hyperplanes, then we have:

$$\mathbb{E}[\frac{1}{n}\sum_{i=1}^{n} X_i] = 1 - \frac{\theta}{\pi} \tag{14}$$

$$\mathbb{V}[\frac{1}{n}\sum_{i=1}^{n} X_i] = \frac{1}{n^2}\sum_{i=1}^{n}\mathbb{V}[X_i] = \frac{1}{n}\cdot\frac{\theta}{\pi}\left(1 - \frac{\theta}{\pi}\right) \tag{15}$$

Set $\overline{X}_n = \frac{1}{n}\sum_{i=1} X_i$. Then we have that $\mathbb{E}[\pi(1 - \overline{X}_n)] = \theta$, suggesting that we use $\cos(\pi(1 - \overline{X}_n))$ as an estimator for $\cos(\theta)$. Let $g(x) = \cos(\pi(1 - x))$ and $C_n = g(\overline{X}_n)$ be this estimator. We now explore some properties of C_n.

Lemma A.1. *C_n is consistent. In particular, $C_n \xrightarrow{a.s.} \cos(\theta)$.*

Proof. By the strong law of large numbers, we have that $\overline{X}_n \xrightarrow{\text{a.s.}} 1 - \frac{\theta}{\pi}$. Since g is continuous, by the continuous mapping theorem [Van Der Vaart, 1998],

$$g(\overline{X}_n) \xrightarrow{\text{a.s.}} g(1 - \frac{\theta}{\pi}) = \cos(\theta)$$

□

Lemma A.2. *$\mathbb{E}[C_n] = \cos(\theta) + E_n$, where $|E_n| \leq \frac{\pi^2}{8n}$*

Proof. Set $\mu = \mathbb{E}[\overline{X}_n] = 1 - \frac{\theta}{\pi}$. The first degree Taylor series for $g(x)$ about μ is:

$$g(x) = \cos(\theta) + \pi\sin(\theta)(x - \mu) + R(x)$$

where R is the remainder term. Applying Lagrange's remainder formula, we have that for each x, there exists some $h(x)$ lying between μ and x such that:

$$R(x) = \frac{\pi^2\cos(\pi h(x))}{2}(x - \mu)^2$$

We have then that:

$$\mathbb{E}[g(\overline{X}_n)] = \cos(\theta) + \pi\sin(\theta)(\mathbb{E}[\overline{X}_n] - \mu) + \mathbb{E}[R(\overline{X}_n)] \tag{16}$$
$$= \cos(\theta) + \mathbb{E}[R(\overline{X}_n)] \tag{17}$$

Thus it suffices to bound $|\mathbb{E}[R(\overline{X}_n)]|$. We have:

$$|\mathbb{E}[R(\overline{X}_n)]| \leq \mathbb{E}[|R(\overline{X}_n)|] \tag{18}$$
$$= \mathbb{E}[|\frac{\pi^2\cos(\pi h(\overline{X}_n))}{2}|(\overline{X}_n - \mu)^2] \tag{19}$$
$$\leq \mathbb{E}[\frac{\pi^2}{2}(\overline{X}_n - \mu)^2] \tag{20}$$
$$= \frac{\pi^2}{2}\mathbb{V}[\overline{X}_n] \tag{21}$$
$$= \frac{\pi^2}{2}\cdot\frac{1}{n}\cdot\frac{\theta}{\pi}(1-\frac{\theta}{\pi}) \tag{22}$$
$$\leq \frac{\pi^2}{2}\cdot\frac{1}{4n} = \frac{\pi^2}{8n} \tag{23}$$

□

Lemma A.3. $\mathbb{V}[C_n] = \frac{\pi^2\sin^2(\theta)}{n}\cdot\frac{\theta}{\pi}(1-\frac{\theta}{\pi}) + O(n^{-3/2})$

Proof. Before delving into the details, let us sketch the idea. The Taylor approximation for g shows us that $g(x) \approx \cos(\theta) + \pi\sin(\theta)(x-\mu)$. So, recalling that $\mathbb{V}[C_n] = \mathbb{V}[g(\overline{X}_n)] = \mathbb{E}[g(\overline{X}_n)^2] - \mathbb{E}[g(\overline{X}_n)]^2$, and plugging in the approximation we get:

$$\mathbb{V}[g(\overline{X}_n)] \approx \mathbb{E}[\big(\cos(\theta) + \pi\sin(\theta)(\overline{X}_n - \mu)\big)^2] - \cos(\theta)^2 \tag{24}$$
$$= 2\pi\sin(\theta)\mathbb{E}[\overline{X}_n - \mu] + \pi^2\sin^2(\theta)\mathbb{E}[(\overline{X}_n - \mu)^2] \tag{25}$$
$$= \pi^2\sin^2(\theta)\mathbb{E}[(\overline{X}_n - \mu)^2] \tag{26}$$

Now, to get the actual error term, we carry out the same process without dropping the remainder term $R(x)$ from the Taylor expansion. We have:

$$\mathbb{V}[g(\overline{X}_n)] = \mathbb{E}[\big(\cos(\theta) + \pi\sin(\theta)(\overline{X}_n - \mu) + R(\overline{X}_n)\big)^2] - \big(\cos(\theta) + \mathbb{E}[R(\overline{X}_n)]\big)^2 \tag{27}$$
$$= \cos(\theta)^2 + 2\cos(\theta)\mathbb{E}[R(\overline{X}_n)] + \mathbb{E}[R(\overline{X}_n)^2] + 2\pi\cos(\theta)\sin(\theta)\mathbb{E}[\overline{X}_n - \mu] \tag{28}$$
$$+ 2\pi\sin(\theta)\mathbb{E}[R(\overline{X}_n)(\overline{X}_n - \mu)] + \pi^2\sin^2(\theta)\mathbb{E}[(\overline{X}_n - \mu)^2] \tag{29}$$
$$- \big(\cos(\theta) + \mathbb{E}[R(\overline{X}_n)]\big)^2 \tag{30}$$
$$= \pi^2\sin^2(\theta)\mathbb{V}[\overline{X}_n] - \mathbb{E}[R(\overline{X}_n)]^2 + \mathbb{E}[R(\overline{X}_n)^2] + 2\pi\sin(\theta)\mathbb{E}[R(\overline{X}_n)(\overline{X}_n - \mu)] \tag{31}$$

It suffices to bound the last three terms. From the previous result, we know that the first of these three terms is the square of something which is $O(1/n)$. For the second,

$$|\mathbb{E}[R(\overline{X}_n)^2]| = \mathbb{E}[R(\overline{X}_n)^2] \tag{32}$$

$$= \mathbb{E}[\frac{\pi^4 \cos^2(\pi h(\overline{X}_n))}{4}(\overline{X}_n - \mu)^4] \tag{33}$$

$$\leq \frac{\pi^4}{4}\mathbb{E}[(\overline{X}_n - \mu)^4] \tag{34}$$

Let $p = \theta/\pi$. Then

$$\mathbb{E}[(\overline{X}_n - \mu)^4] = \frac{3np^4 - 6np^3 + 3np^2 - 6p^4 + 12p^3 - 7p^2 + p}{n^3} \tag{35}$$

$$\leq \frac{6n + 13}{n^3} \tag{36}$$

so that

$$|\mathbb{E}[R(\overline{X}_n)^2]| \leq \frac{\pi^4(6n + 13)}{4n^3} \tag{37}$$

For the final term, we have:

$$|\mathbb{E}[R(\overline{X}_n)(\overline{X}_n - \mu)]| \leq \sqrt{\mathbb{E}[R(\overline{X}_n)^2]\mathbb{E}[(\overline{X}_n - \mu)^2]} \tag{38}$$

$$\leq \sqrt{\frac{\pi^4(6n + 13)}{n^3} \cdot \frac{1}{n} \cdot \frac{\theta}{\pi}(1 - \frac{\theta}{\pi})} \tag{39}$$

$$\leq \sqrt{\frac{\pi^4(6n + 13)}{n^4} \cdot \frac{\theta}{\pi}(1 - \frac{\theta}{\pi})} \tag{40}$$

which is $O(n^{-3/2})$ □

Lemma A.4. $\mathbb{P}(|C_n - \cos(\theta)| > \delta + \frac{\pi^2}{8n}) \leq 2e^{-2n\delta^2/\pi^2}$

Proof. Define $f_n : [0, 1]^n \to \mathbb{R}$ by $f_n(x_1, \ldots, x_n) = \cos(\pi(1 - \frac{1}{n}\sum_{i=1}^n x_i))$. Then $f_n(X_1, \ldots, X_n) = C_n$. We now show that for all i, and $x_1, \ldots, x_n \in [0, 1]^n$ and $x_i' \in [0, 1]$,

$$|f_n(x_1, \ldots, x_i, \ldots, x_n) - f_n(x_1, \ldots, x_i', \ldots, x_n)| \leq \frac{\pi}{n}|x_i - x_i'|$$

Fix i and $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$, and consider the function

$$h(x) = f_n(x_1, \ldots, x_{i-1}, x, \ldots, x_n)$$

Then, by the mean value theorem, for all $x_i, x_i' \in [0, 1]$, there exists some $x^* \in [0, 1]$ such that:

$$h(x_i) - h(x_i') = h'(x^*)(x_i - x_i')$$

$$= \frac{\pi}{n}\sin\left(\pi\left(1 - \frac{1}{n}(x_1 + \cdots + x^* + \cdots + x_n)\right)\right)(x_i - x_i')$$

Hence $|h(x_i) - h(x_i')| \leq \frac{\pi}{n}|x_i - x_i'|$. Since the function f_n is Lipschitz continuous in each coordinate with parameter $\frac{\pi}{n}$, by the method of bounded differences [Dubhashi and Panconesi, 2009, Corollary 5.2], we have:

$$\mathbb{P}(|f_n(X_1, \ldots, X_n) - \mathbb{E}[f_n(X_1, \ldots, X_n)]| > \delta) \leq 2e^{-2n\delta^2/\pi^2}$$

Moreover, from above we have that $|\mathbb{E}[C_n] - \cos(\theta)| < \frac{\pi^2}{8n}$. Hence,

$$\mathbb{P}(|C_n - \cos(\theta)| > \delta + \frac{\pi^2}{8n}) \leq 2e^{-2n\delta^2/\pi^2}$$

□

The above bound is uniform over θ, but stronger bounds for very small or large values of θ can be obtained as follows: Set $p = 1 - \theta/\pi$ and apply standard Chernoff bounds to the binomial variable $\overline{X}_n$. Then observe that if $\overline{X}_n$ deviates from its mean by no more than δ with high probability, then $g(\overline{X}_n)$ is within $|\sup_{p-\delta \leq t \leq p+\delta} g'(t)|\delta$ around $\cos(\theta)$ with at least the same high probability.

Appendix B. Model and parameter estimation

Features Here we provide more details about the features. First recall that we employ two types of feature variables, a "name bag" that represents the features of the authors name and a "context bag" that represents the remaining features in the citation from which the author mention is extracted. We populate the "name" bag of each mention with the full name, the first-initial last name, and character tri-grams of the author's name string as it appears in the mention. We populate the "context" features of each mention with the title and venue, the co-authors with whom the paper is authored, author-provided keywords for the paper. For the title we employ white-space tokenization, character 3-grams and 4-grams, for venue we employ white-space tokenization as well as the entire string and for co-authors we employ the first-initial last-name of all co-authors. Finally for topics, we take the top 3 topics for each citation as determined by a 200-topic LDA model trained on all of DBLP and REXA combined. We employ the training data to experiment with several feature combinations, including using all 200-topics for each citation, but found that this combination of features works best.

For the exact sparse-vector representations, we employ the default implementation in FACTORIE, which employs hashmaps to store the non-zero features and their corresponding weights. For the simhash representation we employ a java implementation of `murmurhash3` to hash each feature to construct the initial dense bit vectors that underly the homomorphic simhash representation of the mentions. These vectors are then added and subtracted during inference to produce the underlying vectors of the entities and subentities on which the actual bits are computed.

Parameter estimation Following previous work, we manually tune the model with exact feature representations on the training data with the help of hyper-parameter search. We use identical parameters for the simhash model, except we lower the translation value for the context bag to compensate for the extra variance that can increase the chance of false-positives. More details follow below.

Model	B3			MUC		
	Precision	Recall	F1	Precision	Recalll	F1
exact	75.3	82.2	78.6	88.3	91.2	89.7
simhash 256	74.3	81.2	77.6	86.5	88.8	87.7
simhash 128	76.1	75.1	75.6	86.8	87.1	86.9
simhash 64	66.0	60.0	62.8	81.2	82.4	81.8
simhash 32	58.0	53.4	55.6	75.8	80.3	78.0
jl 258	73.5	83.9	78.3	87.1	91.2	89.1
jl 128	74.9	77.1	75.9	87.4	87.4	87.4
jl 64	77.2	60.8	68.0	85.7	82.6	84.1
jl 32	74.6	60.0	66.5	84.4	82.6	83.6

Table 1: Author coreference evaluation on REXA with B3 and MUC evaluation measure.

The model comprises only a small handful of parameters: a weight and translation parameter for each bag, and a prior on the number of entities and subentities. Since the prior is somewhat redundant to the translation parameters, we fix the prior to zero and do not tune it. We also fix the parameter for the number of subentities to -0.5 to penalize deeper trees. Then, we tune the remaining parameters manually with the assistance of hyper-parameter search. We explore weight values in the set $\{0.0, 1.0, 2.0, 4.0\}$ and translation values in the set $\{-0.75, -0.5, -0.25, 0.0\}$. For the model that employs exact feature representations, we found a weight of 4.0 and translation of -0.5 works best on the training data for the model that employs exact feature representations. Since the variance of the Simhash estimator increases with the angles, false-positives become a potential issue. Therefore, to compensate for this extra variance, we set the translation for the context bag of the simhash model more aggresively to -0.6 via binary search with the training data and 128-bit model (the other simhash models were not further tuned and employ the same parameters as the 128-bit model).

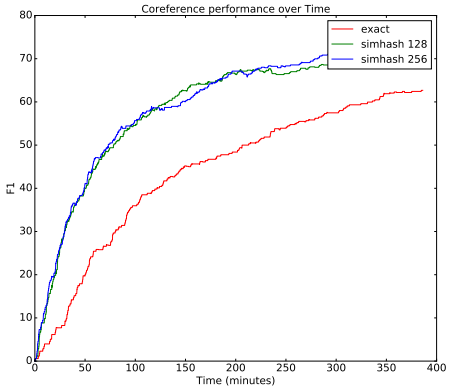
Appendix C. Detailed coreference results

C.1 Evaluation on REXA

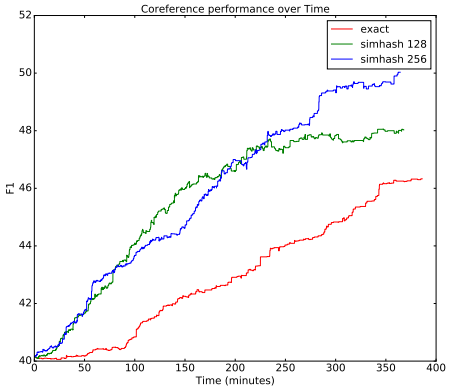
Table 1 contains a more detailed set of results on the REXA author coreference data. We evaluate with both B-cubed (B3) and MUC precision, recall and F1. The results in this table employ all the features of the author mention. For feature ablations, see Table 2 in the sequel.

C.2 Feature ablations

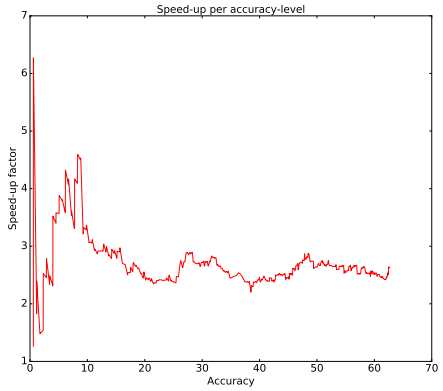
As we speculate in Section 4, we might also be able to improve the speed of the exact model by including fewer features since this would reduce the cost of dynamically updating the sparse-vector representations. To this end, we perform some feature ablations by trying various subsets of the six feature-types: name, topic, co-authors, venue, keywords, and title (with and without n-grams), re-tuning the model on the training set and then reporting the accuracy on the test-set. Note that for the exact model, changing the number of features



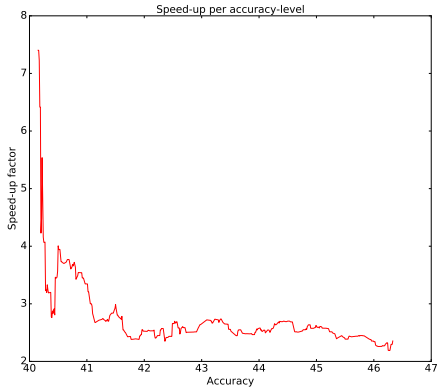
(a) MUC F1 vs time



(b) B3 F1 vs time



(c) Simhash speedup (MUC)



(d) Simhash speedup (B3)

Figure 3: Comparison of the hierarchical coreference models that employ either simhash or exact sparse-vector representations of the features. We removed the n-gram features to improve the run-time of the exact representation on the larger dataset, but this comes at the expense of lower accuracy.

Features	Sample/sec	Precision	Recall	F1 (B-cubed)
name	16750	77.0	44.6	56.5
+topics	8733	75.6	57.5	65.3
+co-authors	1269	75.7	52.3	61.9
+venue	7451	48.3	76.6	59.2
+title	1973	73.6	87.4	79.9
+topics+coauth	13262	75.9	56.9	65.0
all but title	6858	71.0	64.8	67.7
all with title (no n-grams)	10309	71.4	85.2	75.9
all	1835	72.6	86.0	78.7
all (simhash 256 bit)	16556	75.2	81.4	78.2
all (simhash 128 bit)	19841	72.6	82.3	77.2
all (simhash 64 bit)	22988	64.7	65.7	65.2
all (simhash 32 bit)	19417	57.4	59.1	58.3

Table 2: Feature ablations on the test set. B-cubed precision (p), recall (r), and F1. We can see that more features help, but come at a price: slower inference (measured as samples per second).

affects both the quality of the model and its speed, but for the simhash model, it only affects the quality since the representation is fixed in dimension. Therefore, in the spirit of the tradeoff, we include all features for simhash and instead vary the number of bits.

For the feature ablation study, we record the samples per second and the B-cubed F1 on the REXA test set and plot these points in Figure 5 with speed (samples per second) on the x-axis and F1 on the y-axis. Points in the upper-left correspond to models that are accurate, but slow, while points in the lower-right correspond to models that are fast, but inaccurate. Points in the upper-right correspond to models that are both fast and accurate. As we can see, varying the number of features for the exact model indeed improves the performance, but it often comes at the expense of accuracy. The simhash models on the other hand (indicated by red triangles) are able to achieve both speed and accuracy, given enough bits. This scatter plot is intended to understand the trade-off space and we omit the labels on the points to reduce clutter. Instead, we provide these details in a separate table (Table 2).

From these ablations we can see that the best speed-accuracy trade-off for the exact model employs all the features except for the title n-grams. Indeed, the title n-grams are the main culprit for slowing down the exact model since they explode the number of features. Thus, we repeat the experiment for Figure 2 (Experiment 2 in Section 4), with this reduced set of features and manage to improve the running time of the exact model to the point at which the simhash model is only 2-3 times faster. In these experiments, the exact model is much faster than when it had to cope with the entire set of features, and achieves around 7000 samples per second before eventually falling to 5000 as the entities get larger.

While reducing the features is a reasonable strategy for improving the inference speed without resorting to simhash, it also has several drawbacks. First, reducing the number of features can have a negative affect on the accuracy, as we see in Figure 5. When run

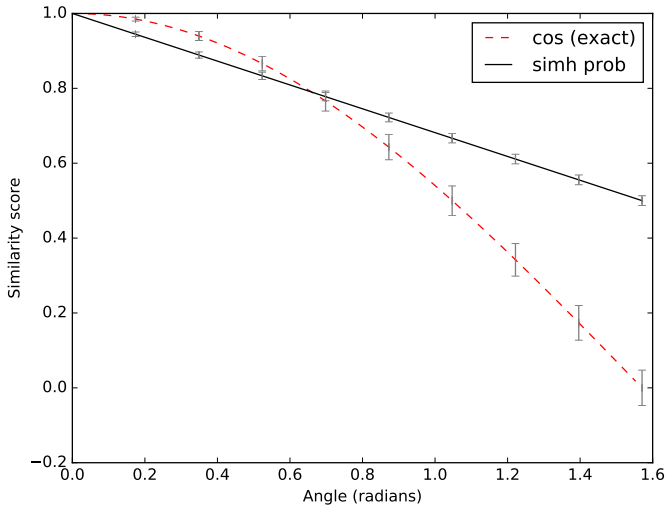


Figure 4: A comparison of exact cosine with two sim-hash approximations. The first is the usual linear approximation, depicted as the linear curve with error bars. The second is as an estimator for cos(*theta*), depicted as the error bars on the exact cosine curve (standard-error of 32-bit hash).

exclusively on the labeled data, the final F1 drops to 75.9, which is lower than the 256-bit simhash model (77.6 F1), while also being twice as slow. In addition, this strategy of reducing the feature space will not work as well in other coreference domains. For example, in author coreference, the true size of author entities is limited by the number of papers the author publishes. It is unlikely that many authors publish more than 1000 papers, and the average is likely to be much smaller. Thus, we can reasonably manage the size of the entity and sub-entity bags simply by reducing the number of features in each mention. In contrast, consider the problem of cross-document coreference on newswire data. The entity "United States" might be mentioned more than a million times in vastly different contexts. Thus, even if we limited the feature-space in the beginning, it is likely that these large entities will still accumulate a large number of features anyway. Models based on simhash will not succumb to this problem. Finally, the simhash representation is appealing because it allows a practitioner to focus on achieving high accuracy with the full flexibility and freedom when engineering features, without having to worry about the running-time performance (an experiment like the type we did in Experiment 1 would help the practitioner set the number of bits without the need for labeled data).

C.3 Note on Memory Use

Our implementation employs 32-bit floats to represent the homomorphic simhash statistics, which results in a memory use of $k * 32$, or between 1kb-8kb per node depending on the number of dimensions. This could be substantially reduced using integer representations

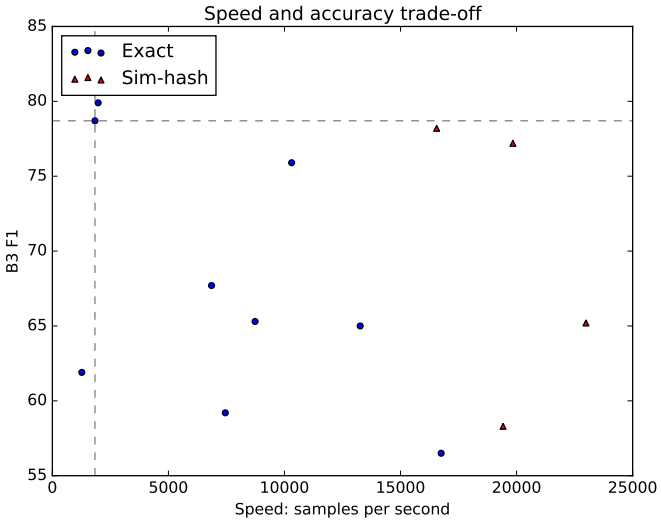


Figure 5: Accuracy versus speed as we vary the number of features for the exact model and number of bits for simhash.

with fewer bits or approximate counters. Although this would still require more memory than non-homomorphic simhash, in the context of hierarchical coreference, the memory use is still small, and often better than the original representation.

Appendix D. Johnson-Lindenstrauss Results

Here we include the details results associated with Experiment 3 in Section 4 in the main paper.

In Figure 6, we compare simhash to the random projection approach, which we abbreviate as JL (for Johnson-Lindenstrauss, since it falls under the purvue of these lemmas). The experiment is analogous to Experiment 1, except that when we run MCMC inference to obtain the data points, we employ the exact model to evaluate the MCMC moves and make acceptance decisions in order to allow for both approximation schemes to be compared on the same set of samples. For accepted sample, we can again ask the question how the approximate methods, simhash and JL, would have judged the MCMC sample. We display the results in a scatter plot with exact model judgements on the x-axis and the approximate method on the y-axis. In the figure, there are eight subplots. The left column contains the simhash comparison and the right column contains the JL comparison. The number of bits increases with each row (32, 54, 128, 256).

In Table 3 we report a numerical comparison to corroborate the scatter plots. In particular, for a fixed number of dimensions, we compare the two methods in terms of the coreference F1 (B-cubed and MUC), their correlation (Spearman's rho) with the exact model, and their accuracy with respect to exact model's decisions in accepting and rejecting

Bits	Algorithm	B3 F1	MUC f1	Spearman's rho	Acc
256	simhash	77.6	87.7	93.2	97.8
	jl	78.3	89.1	97.2	99.3
128	simhash	75.6	86.9	96.4	88.3
	jl	75.9	87.4	95.2	98.3
64	simhash	62.8	81.8	81.2	93.1
	jl	68.0	84.1	90.1	96.9
32	simhash	55.6	78.0	70.0	93.7
	jl	66.5	83.6	81.9	94.6

Table 3: A comparison of SimHash and JL style cosine computations. Spearman's rho and accuracy are computed with respect to the exact cosine computations.

MCMC proposals. We find that a model based on JL cosine estimates are better correlated with the exact model, and this translates into small improvements in coreference F1.

Appendix E. MinHash

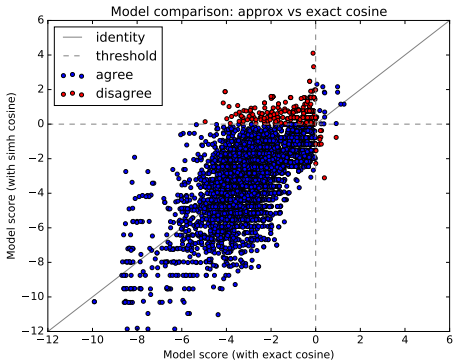
E.1 Homomorphic MinHash

Background: MinHash Minhash [Broder, 1997a, Broder et al., 2000] is a locality-sensitive hash function for Jaccard similarity, defined for binary vectors. Note that binary vectors can also represent sets such that, the non-zero entries correspond to elements in the set. For example, a binary vector in $\mathbb{R}^d$ is equivalent to a set $\Omega = \{0, 1, 2, \ldots, d-1\}$. The Jaccard similarity is a measure of resemblance between two sets $S_1, S_2 \in \Omega$, defined as $J = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$. Minhash applies a random permutation (which can be accomplished using a random hash function e.g., seeded 64-bit murmur hash) $\pi : \Omega \to \Omega$, on a given set $S \subset \Omega$ and then extracts the minimum value $h_\pi(S) = \min(\pi(S))$. The probability that the minhash function for a random permutation of a set computes the same value for two sets is equal to the Jaccard similarity of the two sets ([Rajaraman and Ullman, 2010] illustrates an intuitive derivation). Formally,

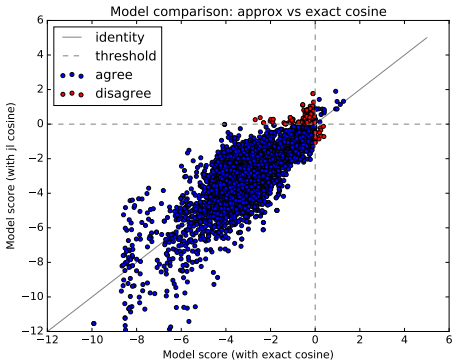
$$Pr(h_\pi(S_1) = h_\pi(S_2)) = \frac{\mid S_1 \cap S_2 \mid}{\mid S_1 \cup S_2 \mid} = J \tag{41}$$

The Jaccard estimate is then given by $\hat{J} = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}\left[h_{\pi_i}(S_1) = h_{\pi_i}(S_2)\right]$. The variance of this estimate is $\text{Var}(\hat{J}) = \frac{1}{n} J(1-J)$.

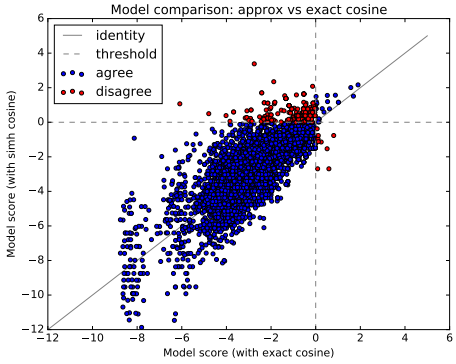
Homomorphic MinHash: Unlike simhash for cosine similarity, the operations underlying the statistics for minhash are non-linear due to the element-wise minimum operation that produces the underlying vector of hash values. Moreover, the set semantics on which minhash relies are problematic because we need to maintain the invariance that a parent set is equal to the union of a collection of child sets: when we perform a difference operation between a parent set and a child set we cannot simply remove all of the child's elements from the parent since a sibling might also (redundantly) contribute some of those elements to the parent.



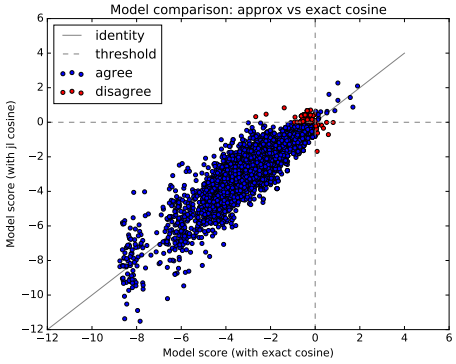
(a) 32-bit



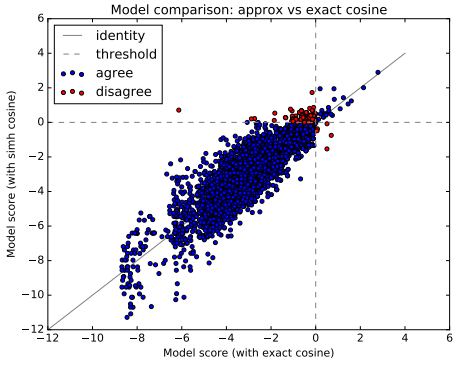
(b) 32-dim



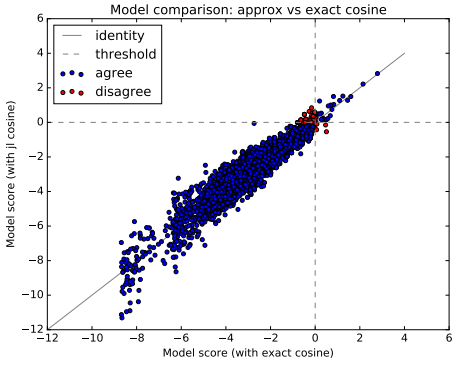
(c) 64-bit



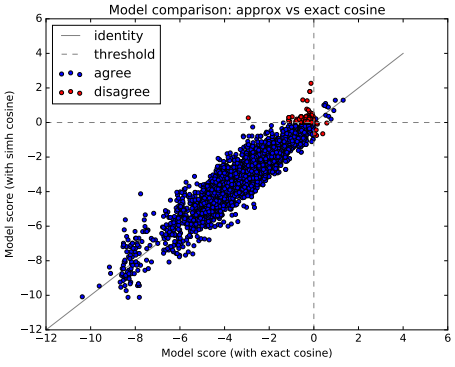
(d) 64-dim



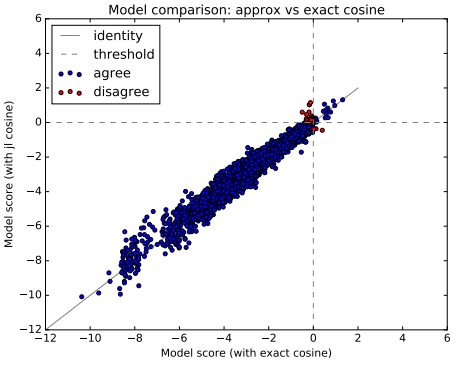
(e) 128-bit



(f) 128-dim



(g) 256-bit



(h) 256-dim

To address this situation, we introduce additional non-negative integer variables as counts that indicate the number of child sets that contribute the associated hash value. We augment the n-dimensional minhash representation v^S of each set S with an n-dimensional vector of counts c^S, in which each dimension corresponds to a minimum hash value. This representation is similar to multiset semantics, but is in the hash space instead.

We observe that given the minhash values of two sets, it is relatively straightforward to compute the minhash value of the union which is simply the minimum of the minhash values of the two nodes being merged, i.e., $h_{\pi_i}(S_1 \cup S_2) = \min(h_{\pi_i}(S_1), h_{\pi_i}(S_2))$. To obtain the corresponding counts, we either keep the count associated with the smaller hash value if they are different (that is, we set $c_i^{S_1 \cup S_2} = c_i^{S^\star}$ where $S^\star = \text{argmin}_{S_j \in \{S_1, S_2\}}(h_{\pi_i}(S_j))$, or sum the counts if they are the same (that is, $c_i^{S_1 \cup S_2} = c_i^{S_1} + c_i^{S_2}$).

In case of set difference however, the computation is not trivial (and as we mentioned earlier, the operation is not the standard set difference either). We consider the following possible cases to illustrate our approach.

(a) $h_{\pi_i}(S_1) < h_{\pi_i}(S_2)$ implies that in the original feature space, there is an element e such that $e \in S_1$ and $e \notin S_2$. In this case, $h_{\pi_i}(S_1 \setminus S_2) = h_{\pi_i}(S_1)$ and the corresponding count is equal to $c_i^{S_1}$.

(b) $h_{\pi_i}(S_1) > h_{\pi_i}(S_2)$ implies that in the original feature space, there is an element e such that $e \in S_2$ and $e \notin S_1$. Again in this case, $h_{\pi_i}(S_1 \setminus S_2) = h_{\pi_i}(S_1)$ and the corresponding count is equal to $c_i^{S_1}$. Note that since we assume that a node being subtracted was previously added to current node at some point, we can eliminate this case.

(c) $h_{\pi_i}(S_1) = h_{\pi_i}(S_2)$ i.e., the minhash values are equal. In this case, if $c_i^{S_1} > c_i^{S_2}$, $h_{\pi_i}(S_1 \setminus S_2) = h_{\pi_i}(S_1)$ and the count is $c_i^{S_1} - c_i^{S_2}$. We do not consider the case $c_i^{S_1} < c_i^{S_2}$ because of the same assumption as above, i.e., a node being subtracted was previously added to current node.

However, when $c_i^{S_1} = c_i^{S_2}$, removing S_2 from S_1 leads to loss of information that can not be retrieved without referring back to the original feature space. Recomputing the minimum from scratch is a nonstarter because it would require keeping around the original sparse vector representations that we are trying to supplant. Rewriting the difference in terms of unions is also computationally expensive since it would require traversing the entire tree to check what the new minimum value should be. Instead, noting that minhash typically has hundreds of hash functions (e.g., $n = 256$) for each set, we propose to ignore the hash values associated with zero counts, and employ the remaining hashes to compute the Jaccard.

This strategy of ignoring hash values that have zero counts has consequences for both bias and variance. First, since fewer hashes are employed, the variance increases, and if left unchecked may culminate in a worst case in which all counts are zero and Jaccard can no longer be estimated. However, we can periodically refresh the counts by traversing the trees and hopefully we do not have to do such refresh operations too often in practice.

Second, since the hashes associated with zero counts are correlated, the estimate is no longer unbiased. For example, consider the case when we merge the node $S_1 = \{\text{mustard}\}$

with the node $S_2 = \{\text{hot-dog, mustard}\}$. Now, if we remove S_2 from $S_1 \cup S_2$, the hash values corresponding to the element "mustard" become zero. Consequently, the differentiating information between $(S_1 \cup S_2) \setminus S_2$ and S_2 is lost resulting in a biased Jaccard estimate, i.e., $\hat{J}((S_1 \cup S_2) \setminus S_2, S_2) = 1$. To address this problem, we modify the Jaccard estimate to make better use of the zero-counts rather than simply ignore them, and this eliminates the bias in some cases. However, we do not have a solution that works in every case.

Finally, there is also the question of how to perform union and difference for cases in which the count is zero. For now, we ignore the zero counts during these operations, but are currently exploring strategies for incorporating them to further reduce bias and variance. Our approach is summarized in Algorithm 1 for the union and difference operations as well as the Jaccard estimate computation (the function `score`).

Algorithm 1 Homomorphic MinHash

function UNION(S_1, S_2)
  $h_{\pi_i}(S_1 \cup S_2) = \min(h_{\pi_i}(S_1), h_{\pi_i}(S_2))$
  if $c_i^{S_1} = c_i^{S_2}$ **then**
    $c_i^{S_1 \cup S_2} = c_i^{S_1} + c_i^{S_2}$
  else
    $c_i^{S_1 \cup S_2} = c_i^{S^\star}$ where $S^\star = \text{argmin}_{S_j \in \{S_1, S_2\}}(h_{\pi_i}(S_j))$

function DIFFERENCE(S_1, S_2)
  if $h_{\pi_i}(S_1) \neq h_{\pi_i}(S_2)$ **then**
    $h_{\pi_i}(S_1 \setminus S_2) = h(S_1), c_i^{S_1 \setminus S_2} = c_i^{S_1}$
  else if $h_{\pi_i}(S_1) = h_{\pi_i}(S_2)$ **then**
    if $c_i^{S_1} > c_i^{S_2}$ **then**
      $h_{\pi_i}(S_1 \setminus S_2) = h(S_1), c_i^{S_1 \setminus S_2} = c_i^{S_1} - c_i^{S_2}$
    else
      $h_{\pi_i}(S_1 \setminus S_2) > h(S_1), c_i^{S_1 \setminus S_2} = 0$

function SCORE(S_1, S_2)
  $\hat{J} = \frac{1}{\hat{n}} \sum_i \mathbb{1}\left[h_{\pi_i}(S_1) = h_{\pi_i}(S_2)\right], \forall i \quad \text{s.t.} \quad c_i^{S_1} > 0 \quad \text{and} \quad c_i^{S_2} > 0$
  $\hat{n}$ is the number of hash function pairs with non-zero counts.

E.2 Homomorphic MinHash Results

Recall that our homomorphic minhash algorithm associates a count with each minimum hash value that indicates the number of times that hash value was a unioned into the set as a minimum. During the difference operation that can occur due to a split proposal in hierarchical coreference, it is possible that these counts can become zero. Then, information is lost because it is not possible to recompute the new minimum value without traversing the entire tree, an operation that is expensive to perform in the inner loop of MCMC inference. Therefore, as a possible remedy, we propose to ignore hash values that have a count of zero in certain situations. In certain situations, this is unbiased[5] A primary concern though,

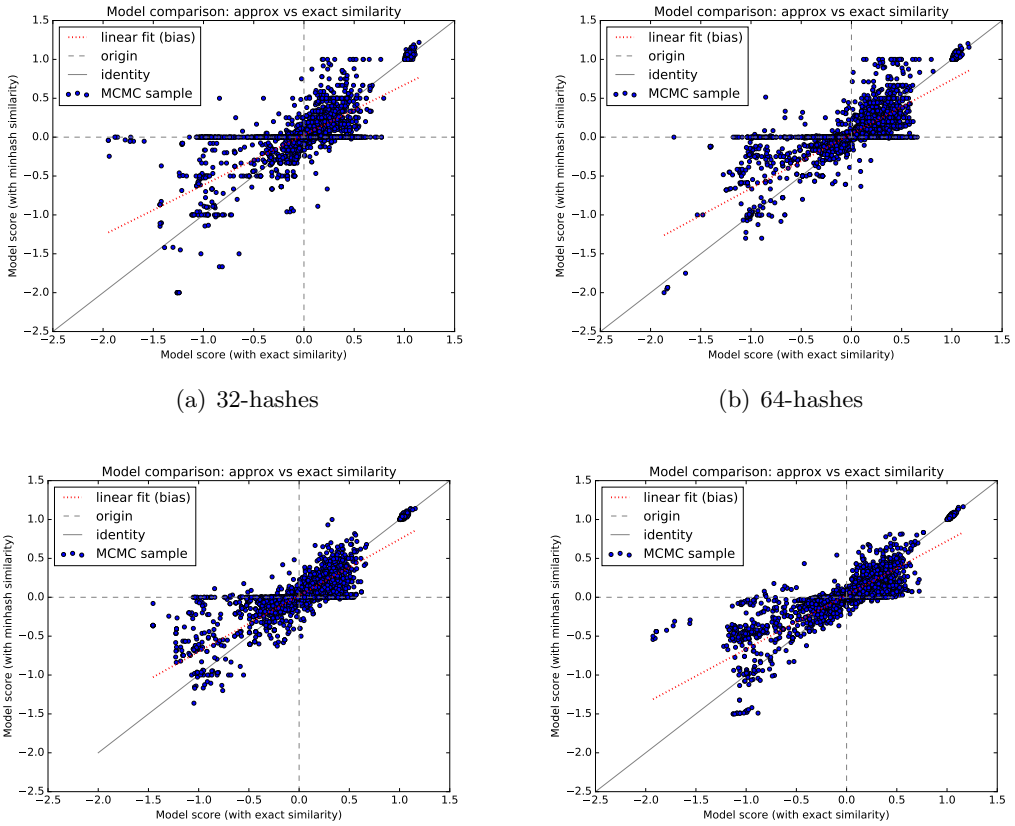
5. in other cases it is biased and we should recompute the values in these cases, but we save this for future work.

is that as the number of samples increases, the number of entries with a zero count will increase and (a) the variance of the estimate might increase and (b) the estimate might be impossible to compute if all counts become zero.

We indirectly study (a) by comparing our homomorphic minhash estimates to the exact Jaccard estimate in much the same we did for simhash (Figure 7). Each point in the plot represents an MCMC sample, which we gather by running the exact model with no compressive data structures as before. The x-axis is the difference in the exact Jaccard similarity estimates of the context bags and the y-axis is the difference in exact Jaccard similarity between the homomorphic minhash representations of these bags. Unlike simhash, minhash employs multiple hash functions; therefore, we fix the number of bits to 64 and vary the number of hash functions instead. We also fit a linear model to the points in order to help assess the bias (as seen by how far off it is from the reference line), and find that indeed our method introduces some bias. We further examine the percentage of hash functions have a non-zero count. In particular, after running hierarchical coref in the previous experiment for 100,000 samples, we look at the root level entities and measure what percentage of their hash functions have counts that have gone to zero). The reason we look at root entities only is that they are the most likely to have had difference operations applied to them since every time MCMC proposes to split a sub-tree, the root must undergo a difference operation. We find that on average about 16% of these hash-values have a non-zero count, and find that 78% or these representations have fewer than 25% non-zeros, 15% have between 25-50% non-zeros, 4% have between 50-75% and the rest have above 75%.

As for (b), we check after each sample if it results in a hash representation for which all its counts are zero, thus making Jaccard estimates impossible.[6] We record for each of the 100,000 samples whether or not the sample results in a hash for which all counts are zero. We find that this only happens for the 32 and 64 hash function variants, and even in these cases, the percentage is small (1.4 and 1.7% respectively), and they are responsible for the regimentation of points along the x-axis in Figures 7(a)–7(b). For 128 and 256 hash function variants, this situation never occurs. This is encouraging as it means, from the perspective of being able to compute an estimate, that we rarely need to refresh the values.

6. Actually, this is not entirely true since some estimates are possible because not all information is lost. Technically a count of zero indicates that although we do not know what the minimum is, we know what it is *not*, viz., the hash value associated with the count. Therefore, if it happens to be the case that the representation we compare against (a) contains non-zeros counts everywhere and (b) has a hash an identical hash-value for each hash function, we can conclude the Jaccard is zero. Our implementation supports this case as a special case, but it is so unlikely to occur.



(a) 32-hashes

(b) 64-hashes

(c) 128-hashes

(d) 256-hashes

Figure 7: Model score comparison with homomorphic minhash and exact sparse-vector representations. The closer points are to the identity reference line, the better.