An incremental data-stream sketch using sparse random projections*

Aditya Krishna Menon† Gia Vinh Anh Pham‡ Sanjay Chawla§ Anastasios Viglas¶

Abstract

We propose the use of random projections with a sparse matrix to maintain a sketch of a collection of high-dimensional data-streams that are updated asynchronously. This sketch allows us to estimate L_2 (Euclidean) distances and dot-products with high accuracy. We verify the validity of this sketch by applying it to an online clustering problem, where we compare our results to the offline algorithm and an existing L_2 sketch, and observe comparable results in terms of accuracy, and a reduced runtime cost.

1 Introduction.

1.1 Data-streams. A data stream, as the metaphor suggests, is the continuous flow of data generated at a source (or multiple sources) and transmitted to various destinations. Most data-stream frameworks for data-mining assume that data arrives incrementally and sequentially (in-order), and develop solutions for this case. But in many situations, the data not only arrives incrementally, but also *asynchronously* - that is, at any point in time, we may be asked to update any arbitrary component or attribute of the existing data. The stream here cannot be mined with a one-pass algorithm, since the update of components is not (necessarily) sequential. Therefore, the challenge is to carry out our data mining task without having to instantiate the data vector - which may not be possible or computationally feasible.

1.2 Problem definition. Consider a collection of n objects, each with dimension d, represented by the $n \times d$ matrix $A = \begin{bmatrix}\mathbf{u_1} & \mathbf{u_2} & \dots & \mathbf{u_n}\end{bmatrix}^T$. Suppose that this matrix is updated asynchronously at various points in time, by updates of the form (i, j, c), which instructs us to update row i, column j with the value c (which may be negative).

Suppose now that the value of d can be so large as to make instantiation of the matrix A infeasible. However, we would still like to be able to extract some useful information about the data; for example, we might want to estimate the L_2 norm (Euclidean distance) of the individual rows/streams of the data, or perhaps the dot-product between streams.

To make such estimates, a *sketch* of the streams is usually kept. A sketch is some approximation of a stream (or a series of streams) that has two desirable properties: firstly, it occupies very little space, and secondly, it allows us to give quick and accurate answers to queries on some quantity associated with the stream.

1.3 Our contribution. We propose the use of random projections with a sparse matrix, defined by Achlioptas [1], to maintain a sketch of a collection of high-dimensional streams that are updated asynchronously. This sketch allows us to estimate L_2 (Euclidean) distances and dot-products with high accuracy. The advantages of this approach are simplicity, and efficiency.

Further, we show how the results on L_2 preservation allow us to estimate dot-products as well, and derive bounds on the maximal error of our estimate. We also derive a similar result for an existing L_2 sketch.

We verify the validity of our projection-based sketch by applying it to an online clustering problem, where we have to cluster high-dimensional data streams that are updated incrementally at some given points in time. We compare our results to the offline algorithm and an existing L_2 sketch, and observe comparable results in terms of accuracy, and an improved runtime.

1.4 Related work.

1.4.1 L_2-approximating sketches. An important sketch for preserving distances of streams is the work of Indyk [6], who proposed the use of a p-stable distribution in order to estimate the L_p norm, $||\mathbf{x}||_p := \left(\sum_i |x_i|^p\right)^{1/p}$, for arbitrary $p \in (0, 2]$. For $p = 2$, this requires the use of Gaussian random variables to maintain a sketch, and so the method used is essentially a random projection, with some minor differences; we expand on this in §4.

*Sanjay Chawla was partially funded by ARC DP0559005. Anastasios Viglas was partially funded by ARC DP0664782.

†The University Of Sydney.
‡The University Of Sydney.
§The University Of Sydney.
¶The University Of Sydney.

1.4.2 Clustering.

[5] provides one-pass algorithm for clustering a set of points; this however does not generalize to clustering an asynchronous stream of data, where we can update existing points.

Random projections were used to cluster non-streaming data in [3], where it was suggested that an ensemble of projections be used for best results. Such an approach would be a natural extension to our work.

2 Background.

Random projections are a powerful method of dimensionality reduction that have been applied in numerous practical problems [2, 9, 4], and have also served as a useful tool in algorithmic design [11]. The basis of projections is the remarkable Johnson-Lindenstrauss lemma [7], which says that for any data set of n points in d dimensions, and some error parameter ϵ, it is possible to find a mapping $f : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times k}$, where $k = O\left(\log n/\epsilon^2\right)$, such that all pairwise distances between points are preserved within a factor of ϵ. Since the proof of the lemma did not specify how one would find such an f in general, in practise, one tends to use approximate Johnson-Lindenstrauss embeddings, which give the same guarantees, but probabilistically.

We can think of projections as a multiplication by a suitably chosen random matrix:

$$f : A \mapsto A.R$$

where R is a matrix whose entries follow some random distribution. The "classic" form of random projections used a matrix with entries being i.i.d. $\mathcal{N}(0, 1)$. Achlioptas [1] showed that a much simpler distribution,

$$r_{ij} = \begin{cases} +\sqrt{3} & p = 1/6 \\ 0 & p = 2/3 \\ -\sqrt{3} & p = 1/6 \end{cases}$$

suffices. We henceforth call this matrix the Achlioptas matrix. This matrix has an expected sparsity of $\frac{2}{3}$rds, making it quite desirable to use in practise[1].

3 Random projections and streams.

3.1 Projection-based sketch.

Suppose we have a data matrix $A(t) \in \mathbb{R}^{n\times d}$, which is updated at arbitrary points in time. Now, suppose that updates happen to this matrix in the following way. At time t, we get a tuple $d(t) = (i_t, j_t, c_t)$. When this happens,

$$a_{i_t j_t}(t) = a_{i_t j_t}(t-1) + c_t$$

That is, the entry in cell (i_t, j_t) gets incremented by the quantity c_t. We can express this as

$$A(t) = A(t-1) + C(t)$$

where $C(t) = \{C_{mn}\}$, such that $C_{mn} = c_t \delta_{i_t m} \delta_{j_t n}$ (where δ denotes the Kronecker delta function).

Suppose that initially, $A(0) \equiv \mathbf{0}$ - we have no information about any component at all. It is clear then that, based on the nature of the updates, we must have

$$A(t) = \sum_{n=1}^{t} C(n) \tag{3.1}$$

If we keep a sketch $E(t)$ of the matrix $A(t)$ using a random projection, then we will have

$$E(t) = \frac{1}{\sqrt{k}} A(t) R^* := A(t) R$$

where R^* is, say, the Achlioptas matrix.

Equivalently, by Equation 3.1,

$$E(t) = \sum_{n=1}^{t} C(n) R$$

What happens the next time we get a tuple? Ideally, we would like to use the existing $E(t)$ matrix, and update it rather than doing a full recomputation. It is easy to see the recurrence:

$$\begin{aligned} E(t+1) &= \sum_{n=1}^{t+1} C(n) R \\ &= E(t) + C(t+1) R \end{aligned}$$

So, the update to the sketch is very simple - we project the matrix $C(t+1)$, and add this to our existing sketch. But of course, $R \in \mathbb{R}^{d\times k}$, which is comparable with A in terms of size. So, we can't really store the random matrix R and use it for each of our projections. We deal with this problem in the next section.

3.2 Generation of R.

3.2.1 Multiplication by C.

The fact that there are many zeros in the C matrix naturally suggests that we are saved a lot of work, and this is precisely the case:

$$\begin{aligned} C(t).R &= \begin{bmatrix} \mathbf{0} & \dots & \mathbf{c_t} & \dots & \mathbf{0} \end{bmatrix}^T \begin{bmatrix} \mathbf{r_1} & \mathbf{r_2} & \dots & \mathbf{r_k} \end{bmatrix} \\ &= c_t \begin{bmatrix} 0 & 0 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots \\ r_{j_t 1} & r_{j_t 2} & \dots & r_{j_t k} \\ \vdots & \vdots & \dots & \vdots \\ 0 & 0 & \dots & 0 \end{bmatrix} \end{aligned}$$

[1]Recently, an even sparser matrix has been proposed in [8], which can give a $\sqrt{d}$-fold speedup in computations

where $\mathbf{c_t} = [0 \quad \ldots \quad c_t \quad \ldots 0]$. So, for each update, we only need to generate k random numbers $r_{j_t i}$, as the rest cancel. This is not particularly difficult; however, the problem is that we want to generate the *same* random numbers each time we refer to the same row.

3.2.2 Pseudo-random generator.

To get repeatable generation of the random rows, we use a pseudo-random number generator (PRNG) for generating the row values. Suppose that row i is generated using a seed $s(i)$ - then certainly if we fix $s(i)$ to be constant at all times, we can be guaranteed to get the same random row each time. However, since there are d rows, due to space constraints we cannot store d seeds, one for each row.

But we can get around this in a very simple way, because the seed can be set to be the original column index, or the random row index, j_t [6]. With this seed, the PRNG generates for us k values corresponding to the random row j_t. Since at times t_1 and t_2 where we refer to the same row, $j_{t_1} \equiv j_{t_2}$ trivially, we will generate the same random row. So, with a PRNG, we can easily generate the rows of R with repeatability, thus solving the problem of not being able to store R.

3.3 Complexity and comments.

By the results of 3.1 and 3.2.2, we see that using random projections, we can keep a sketch with space $\Theta(nk)$ that serves as an approximation to the original matrix $A(t)$, and this sketch can be updated in $\Theta(k)$ time, with no extra space overhead. By the standard projection guarantees, for $k = O(\log n/\epsilon^2)$, we can answer distance queries on $E(t)$ that are accurate up to a factor $(1 \pm \epsilon)$ with high probability.

3.4 The preservation of dot-product.

Random projections also let us get useful guarantees on the preservation of dot-products. It has been shown in e.g. [8] that dot-products are on average preserved under projections, but with a high variance in the answer. We are able to derive the following result on the error incurred in the dot-product estimate.

THEOREM 3.1. *Let $\mathbf{v_i}$ denote the projection of $\mathbf{u_i}$. With high probability, we have that*

$$|\mathbf{v_i}.\mathbf{v_j} - \mathbf{u_i}.\mathbf{u_j}| \leq \frac{\epsilon}{2}(||\mathbf{u_i}||^2 + ||\mathbf{u_j}||^2)$$

Proof. To see this, we use the fact that

$$\mathbf{x}.\mathbf{y} = \frac{||\mathbf{x}+\mathbf{y}||^2 - ||\mathbf{x}-\mathbf{y}||^2}{4} \tag{3.2}$$

By the standard projection guarantees (e.g. [1]), we know that, with high probability,

$$(1-\epsilon)||\mathbf{u_i} - \mathbf{u_j}||^2 \leq ||\mathbf{v_i} - \mathbf{v_j}||^2 \leq (1+\epsilon)||\mathbf{u_i} - \mathbf{u_j}||^2$$

It can be shown that (see [10]), with high probability

$$(1-\epsilon)||\mathbf{u_i} + \mathbf{u_j}||^2 \leq ||\mathbf{v_i} + \mathbf{v_j}||^2 \leq (1+\epsilon)||\mathbf{u_i} + \mathbf{u_j}||^2$$

Dividing by 4 and subtracting the two equations, by Equation 3.2,

$$\mathbf{u_i}.\mathbf{u_j} - \frac{\epsilon}{4}(||\mathbf{u_i} - \mathbf{u_j}||^2 + ||\mathbf{u_i} + \mathbf{u_j}||^2) \leq \mathbf{v_i}.\mathbf{v_j} \leq$$
$$\mathbf{u_i}.\mathbf{u_j} + \frac{\epsilon}{4}(||\mathbf{u_i} - \mathbf{u_j}||^2 + ||\mathbf{u_i} + \mathbf{u_j}||^2)$$

From here, we can say

$$\begin{aligned}|\mathbf{v_i}.\mathbf{v_j} - \mathbf{u_i}.\mathbf{u_j}| &\leq \frac{\epsilon}{4}(||\mathbf{u_i} - \mathbf{u_j}||^2 + ||\mathbf{u_i} + \mathbf{u_j}||^2) \\ &= \frac{\epsilon}{2}(||\mathbf{u_i}||^2 + ||\mathbf{u_j}||^2)\end{aligned}$$

So, we see that with high probability,

$$|\mathbf{v_i}.\mathbf{v_j} - \mathbf{u_i}.\mathbf{u_j}| \leq \frac{\epsilon}{2}(||\mathbf{u_i}||^2 + ||\mathbf{u_j}||^2)$$

4 Comparison with L_p sketch.

4.1 The L_p sketch.

In [6], it was proposed to use a p-stable distribution in order to estimate the L_p norm, $||\mathbf{x}||_p := \left(\sum_i |x_i|^p\right)^{1/p}$. In particular, [6] deals with the problem of a single vector $\mathbf{a}$ that gets updates of the form (i_t, a_t) at time t, indicating that index i_t should be updated by the value a_t. For the case $p = 2$, where we are preserving L_2 (Euclidean) distance, we can interpret the sketch as a projection as stated below.

THEOREM 4.1. *For the case $p = 2$, the L_p sketch can be considered a projection using the classical normal projection matrix $X = [\mathbf{x_1} \quad \mathbf{x_2} \quad \ldots \quad \mathbf{x_k}]$, where $\mathbf{x_i}$ is a $d \times 1$ column vector, the entries $x_{ij} \sim \mathcal{N}(0, 1)$.*

However, in contrast to the projection sketch, there is no scaling of the matrix X by $\frac{1}{\sqrt{k}}$, and further, the estimate for the norm is given by

$$est(||\mathbf{a}||^2) = \frac{median(s_1^2, s_2^2, \ldots, s_k^2)}{median(|Z|^2)}$$

where $Z \sim \mathcal{N}(0, 1)$.

Proof. See [10].

So, the main difference in using the L_p sketch is that we need to generate Gaussian random variables in order to preserve distances. Therefore, our insight is that this expensive Gaussian matrix generation can be replaced by the more efficient generation of a $\frac{2}{3}$rds sparse matrix, without any sacrifice in the accuracy of sketch.

4.2 Bounds on dot-product estimate. We can use the L_p sketch to get the following dot-product estimate.

THEOREM 4.2. *Define*

$$est(\mathbf{u_i}.\mathbf{u_j}) := (1-\epsilon^2)\frac{est(||\mathbf{u_i}+\mathbf{u_j}||^2) - est(||\mathbf{u_i}-\mathbf{u_j}||^2)}{4}$$

Then, with high-probability, we have

$$|est(\mathbf{u_i}.\mathbf{u_j}) - \mathbf{u_i}.\mathbf{u_j}| \leq \frac{\epsilon}{2}(||\mathbf{u_i}||^2 + ||\mathbf{u_j}||^2)$$

Proof. See [10].

Note that this is the same error bound as with projections (Theorem 3.1); therefore, for both sketches, we have the same high-probability upper bound on the error incurred using a simple dot-product estimate.

5 Experiments.

5.1 Time for generation of Gaussian variables. It is intuitive that the generation of a Gaussian random variable would, on average, take longer than the generation of a uniform random variable. Of course, the precise difference depends a lot on implementation. We ran tests comparing the time taken for both on a Pentium-D 3.2 GHz machine with 3.5 GB of RAM, presented in Table 1. Our results on MATLAB indicated that, surprisingly, the generation of Gaussian variables via the built-in `randn` function was faster than the generation of uniform random variables via the built-in `rand`. We also tried the GNU Scientific Library[2] (GSL) for C, and compared the times taken for its implementations. Our results here indicated that uniform random generation was on average faster than the generation of a Gaussian.

5.2 Clustering test. We looked at the quality of the solution generated by random projections by applying the idea to the clustering of data streams. The scenario is that of §1.2; namely, we have n high-dimensional streams that are updated asynchronously, and we would like to cluster these streams at various points in time. We keep a low-dimensional sketch of the streams that preserves Euclidean distances, with the intent that a clustering of the sketch corresponds to a good clustering of the streams. So, we tried clustering using both our projection-based sketch with Achlioptas' matrix (see §2), and the L_2 sketch of Indyk.

We ran experiments (using MATLAB, on a Pentium-D 3.2 GHz machine with 3.5 GB of RAM) for a randomly generated data-set with updates (see §5.2.1) and used the k-means and kernel k-means clustering algorithm to cluster the data. The latter uses dot-products to measure distance, and the former Euclidean distances. The reduced dimension k was varied, and we observed the effect on the solution quality. We did not follow Achlioptas' bound for k (Theorem 2 in [1]) because these bounds are known to be weak in practise [2, 9, 8].

5.2.1 Data-set. For our data-set, we generated a Gaussian mixture, consisting of m cluster centres (where m was made be to either 2 or 5), and filled with n points in d dimensions. The centres of the clusters were chosen randomly, and we used a mean cluster spread of $\sigma^2 = 9$.

We then generated a new mixture, and for each point in the original clustering, chose a random point in the new clustering as its "partner". The updates were chosen to make the original points transform into the partner points, one dimension at a time (so, in total, there were nd updates in total).

We clustered the data periodically, with a total of 50 clusterings. The desired number of clusters in our clustering step was m, the true number of clusters in the data-set, so that the offline algorithm would be expected to give excellent results, which the streaming algorithms could be compared to.

5.2.2 Measuring cluster quality. To assess the cluster quality, we used two metrics - a similarity metric, and an intra-cluster centroid distance metric.

The first measure, which we henceforth refer to as the *similarity* of two clusterings, finds how many pairs of points lie in the same/different cluster for two given clusterings. This is robust against relabellings of clusters, and so is useful for k-means clustering (which on two runs can produce differently labelled, though actually identical clusterings). We computed the similarity of the online (sketch-based) clustering with respect to the offline clustering.

The other measure was the distance of points to the centroids of the clusters they are in. Suppose we have m clusters $c_1, \ldots, c_m$ of our points $x_1, \ldots, x_n$. These clusters are defined by centroids, $r_1, \ldots, r_m$. The objective of k-means clustering is to minimize $\sum_{i=1}^{m} \sum_{x_j \in c_i} ||\mathbf{x_j} - \mathbf{r_i}||^2$. A natural measure of cluster quality is therefore the value of this function - the smaller the value, the "better" the clustering. We computed the ratio of the centroid sum of the online clustering to that of the offline clustering.

5.2.3 Results. Our results for k-means and kernel k-means clustering are presented in Tables 2 and 3. We can see that as the reduced dimension k increases, we

[2] `http://www.gnu.org/software/gsl/`

	MATLAB		GSL	
# values	Uniform	Gaussian	Uniform	Gaussian
10^5	0.006074 ± 0.000050	0.004282 ± 0.000056	0.0043 ± 0.0049	0.0016 ± 0.0050
10^6	0.062306 ± 0.000252	0.044472 ± 0.000120	0.0441 ± 0.0048	0.1572 ± 0.0045
10^7	0.625017 ± 0.006305	0.441509 ± 0.000677	0.4726 ± 0.0267	1.6406 ± 0.0945
10^8	6.22510 ± 0.017631	4.415166 ± 0.022836	4.4054 ± 0.1700	17.8785 ± 0.5173

Table 1: Average time taken and standard deviation, in seconds, to generate uniform and Gaussian random variables over 10,000 runs, with MATLAB and GSL

usually get better results in terms of clustering quality. This is as expected, since the higher the dimension we project to, the less error we incur. We also see that in general, the results for kernel k-means are not as accurate as those for k-means. This is also as expected, as the error incurred in the dot-product does not have as tight a bound as there is for distance.

We also note that projections, on average, manage to out-perform the L_2 sketch in terms of quality of clustering, using either measure.

We measured the time taken for the update of the sketch, which essentially involves the multiplication of a random row by the update value (not including the generation of the random row itself). We varied the number of reduced dimensions k, and tried a large volume of updates of each of the sketches. Our results are presented in Table 4, and indicate that the update time for the projection sketch was faster than that of the L_2 sketch, which is to be expected, since our sketch involves a $\frac{2}{3}$rds sparse row, allowing for a reduction in the number of multiplications that are required.

6 Conclusion.

We propose the use of random projections with a sparse matrix in data-streaming problems, as an extension of the L_2 sketch outlined by [6]. We test the quality of the sketch provided by sparse-projections by applying the sketch to an online clustering problem, and our results indicate comparable accuracy to the earlier L_2 sketch, and an improved runtime performance.

References

[1] Dimitris Achlioptas. Database-friendly random projections. In *PODS '01: Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 274–281, New York, NY, USA, 2001. ACM Press.

[2] Ella Bingham and Heikki Mannila. Random projection in dimensionality reduction: applications to image and text data. In *KDD '01: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 245–250, New York, NY, USA, 2001. ACM Press.

[3] Xiaoli Fern and Carla Brodley. Random projection for high dimensional data clustering: A cluster ensemble approach. In *The Twentieth International Conference on Machine Learning (ICML-2003)*, August 2003.

[4] Dmitriy Fradkin and David Madigan. Experiments with random projections for machine learning. In *KDD '03: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 517–522, New York, NY, USA, 2003. ACM Press.

[5] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *FOCS '00: Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, page 359, Washington, DC, USA, 2000. IEEE Computer Society.

[6] Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *J. ACM*, 53(3):307–323, 2006.

[7] W.B. Johnson and J. Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. In *Conference in Modern Analysis and Probability*, pages 189–206, Providence, RI, USA, 1984. American Mathematical Society.

[8] Ping Li, Trevor J. Hastie, and Kenneth W. Church. Very sparse random projections. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 287–296, New York, NY, USA, 2006. ACM Press.

[9] Jessica Lin and Dimitrios Gunopulos. Dimensionality reduction by random projection and latent semantic indexing. In *Proceedings of the Text Mining Workshop, at the 3rd SIAM International Conference on Data Mining*, 2003.

[10] Aditya Krishna Menon, Anh Pham, Sanjay Chawla, and Anastasios Viglas. An incremental data-stream sketch using sparse random projections. Technical Report 609, The University Of Sydney, January 2007. Available at `http://www.it.usyd.edu.au/research/tr/tr609.pdf`.

[11] Santosh S. Vempala. *The Random Projection Method*, volume 65 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, Providence, RI, USA, 2004.

	Variables			Quality measure			
				Similarity (%)		Centroid ratio	
	n	m	k	Projections	L_2	Projections	L_2
k-means	100	2	5	76.19 ± 20.72	67.88 ± 19.28	0.95 ± 0.15	0.87 ± 0.19
	100	2	100	92.82 ± 17.03	88.09 ± 19.12	1.01 ± 0.02	0.99 ± 0.02
	100	2	200	91.35 ± 18.99	89.25 ± 19.70	1.01 ± 0.08	1.01 ± 0.08
	100	5	5	71.04 ± 9.21	68.49 ± 6.88	0.89 ± 0.30	0.81 ± 0.20
	100	5	100	84.82 ± 9.48	80.36 ± 7.98	1.00 ± 0.41	0.99 ± 0.53
	100	5	200	87.78 ± 9.56	83.20 ± 9.01	1.00 ± 0.23	1.00 ± 0.89
	1000	2	5	84.87 ± 16.92	77.22 ± 18.01	0.95 ± 0.03	0.89 ± 0.06
	1000	2	100	99.15 ± 3.15	93.27 ± 13.75	0.99 ± 0.01	0.99 ± 0.01
	1000	2	200	99.72 ± 1.27	95.65 ± 11.78	0.99 ± 0.01	0.99 ± 0.01
	1000	5	5	71.18 ± 10.91	75.57 ± 10.10	0.81 ± 0.25	0.84 ± 0.51
	1000	5	100	93.06 ± 6.99	89.25 ± 10.02	1.00 ± 0.33	0.98 ± 0.24
	1000	5	200	94.18 ± 5.62	92.67 ± 9.68	0.99 ± 0.26	0.99 ± 0.21
kernel k-means	100	2	5	78.70 ± 19.83	70.74 ± 15.11	0.93 ± 0.06	0.82 ± 0.13
	100	2	100	92.34 ± 17.92	68.92 ± 16.65	1.01 ± 0.06	0.84 ± 0.15
	100	2	200	92.59 ± 17.64	70.21 ± 16.19	1.04 ± 0.26	0.87 ± 0.35
	100	5	5	68.08 ± 7.37	66.72 ± 2.70	0.90 ± 0.23	0.73 ± 0.19
	100	5	100	74.30 ± 10.73	66.36 ± 3.69	1.00 ± 0.13	0.76 ± 0.18
	100	5	200	76.30 ± 9.57	66.66 ± 3.99	1.00 ± 0.35	0.74 ± 0.24
	1000	2	5	74.32 ± 18.80	75.36 ± 17.93	0.87 ± 0.07	0.87 ± 0.07
	1000	2	100	98.89 ± 4.34	75.86 ± 17.26	0.99 ± 0.01	0.87 ± 0.07
	1000	2	200	99.64 ± 1.60	76.67 ± 17.34	0.99 ± 0.01	0.88 ± 0.07
	1000	5	5	60.79 ± 11.19	49.44 ± 12.12	0.94 ± 0.07	0.71 ± 0.25
	1000	5	100	72.71 ± 12.53	55.81 ± 11.77	0.99 ± 0.11	0.70 ± 0.24
	1000	5	200	74.44 ± 12.66	51.94 ± 12.13	0.99 ± 0.13	0.71 ± 0.24

Table 2: Average similarity and centroid radio with standard deviation

	10^4 updates		10^5 updates		10^6 updates	
k	Projections	L_2	Projections	L_2	Projections	L_2
5	0.0588 ± 0.0007	0.0594 ± 0.0006	0.6061 ± 0.0037	0.5977 ± 0.0036	5.8064 ± 0.0066	5.8430 ± 0.0146
100	0.0772 ± 0.0006	0.0870 ± 0.0007	0.8055 ± 0.0048	0.8623 ± 0.0046	7.7333 ± 0.0119	8.6419 ± 0.0311
200	0.0981 ± 0.0007	0.1096 ± 0.0005	1.0476 ± 0.0079	1.0928 ± 0.0058	10.6300 ± 0.0353	10.6701 ± 0.0513
300	0.1195 ± 0.0009	0.1265 ± 0.0007	1.2738 ± 0.0047	1.2683 ± 0.0057	11.7557 ± 0.0575	12.5707 ± 0.0777
400	0.1312 ± 0.0019	0.1438 ± 0.0013	1.4301 ± 0.0051	1.4159 ± 0.0074	13.0989 ± 0.0941	14.1790 ± 0.0220
500	0.1580 ± 0.0007	0.1608 ± 0.0012	1.4601 ± 0.0040	1.5827 ± 0.0035	15.6372 ± 0.0197	15.9051 ± 0.0276

Table 3: Average time and standard deviation, in seconds, for sketch updates