COMPUTER SCIENCE

A neural algorithm for a fundamental computing problem

Sanjoy Dasgupta,[1] Charles F. Stevens,[2,3] Saket Navlakha[4]*

Similarity search—for example, identifying similar images in a database or similar documents on the web—is a fundamental computing problem faced by large-scale information retrieval systems. We discovered that the fruit fly olfactory circuit solves this problem with a variant of a computer science algorithm (called locality-sensitive hashing). The fly circuit assigns similar neural activity patterns to similar odors, so that behaviors learned from one odor can be applied when a similar odor is experienced. The fly algorithm, however, uses three computational strategies that depart from traditional approaches. These strategies can be translated to improve the performance of computational similarity searches. This perspective helps illuminate the logic supporting an important sensory function and provides a conceptually new algorithm for solving a fundamental computational problem.

An essential task of many neural circuits is to generate neural activity patterns in response to input stimuli, so that different inputs can be specifically identified. We studied the circuit used to process odors in the fruit fly olfactory system and uncovered computational strategies for solving a fundamental machine learning problem: approximate similarity (or nearest-neighbors) search.

The fly olfactory circuit generates a "tag" for each odor, which is a set of neurons that fire when that odor is presented (*1*). This tag is critical for learning behavioral responses to different odors (*2*). For example, if a reward (e.g., sugar water) or a punishment (e.g., electric shock) is associated with an odor, that odor becomes attractive (a fly will approach the odor) or repulsive (a fly will avoid the odor), respectively. The tags assigned to odors are sparse—only a small fraction of the neurons that receive odor information respond to each odor (*3*–*5*)—and nonoverlapping: Tags for two randomly selected odors share few, if any, active neurons, so that different odors can be easily distinguished (*1*).

The tag for an odor is computed by a three-step procedure (Fig. 1A). The first step involves feedforward connections from odorant receptor neurons (ORNs) in the fly's nose to projection neurons (PNs) in structures called glomeruli. There are 50 ORN types, each with a different sensitivity and selectivity for different odors. Thus, each input odor has a location in a 50-dimensional space determined by the 50 ORN firing rates. For each odor, the distribution of ORN firing rates across the 50 ORN types is exponential, with a mean that depends on the concentration of the odor (*6*, *7*). For the PNs, this concentration dependence is removed (*7*, *8*); that is, the distribution of firing rates across the 50 PN types is exponential, with close to the same mean for all odors and all odor concentrations (*1*). Thus, the first step in the circuit essentially "centers the mean"—a standard preprocessing step in many computational pipelines—using a technique called divisive normalization (*8*). This step is important so that the fly does not mix up odor intensity with odor type.

The second step, where the main algorithmic insight begins, involves a 40-fold expansion in the number of neurons: Fifty PNs project to 2000 Kenyon cells (KCs), connected by a sparse, binary random connection matrix (*9*). Each KC receives and sums the firing rates from about six randomly selected PNs (*9*). The third step involves a winner-take-all (WTA) circuit in which strong inhibitory feedback comes from a single inhibitory neuron, called APL (anterior paired lateral neuron). As a result, all but the highest-firing 5% of KCs are silenced (*1*, *3*, *4*). The firing rates of these remaining 5% correspond to the tag assigned to the input odor.

From a computer science perspective, we view the fly's circuit as a hash function, whose input is an odor and whose output is a tag (called a hash) for that odor. Although tags should discriminate odors, it is also to the fly's advantage to associate very similar odors with similar tags (Fig. 1B), so that conditioned responses learned for one odor can be applied when a very similar odor, or a noisy version of the learned odor, is experienced. This led us to conjecture that the fly's circuit produces tags that are locality-sensitive; that is, the more similar a pair of odors (as defined by the 50 ORN firing rates for that odor), the more similar their assigned tags. Locality-sensitive hash [LSH (*10*, *11*)] functions serve as the foundation for solving numerous similarity search problems in computer science. We translated insights from the fly's circuit to develop a class of LSH algorithms for efficiently finding approximate nearest neighbors of high-dimensional points.

Imagine that you are provided an image of an elephant and seek to find the 100 images—out of the billions of images on the web—that look most similar to your elephant image. This is called the nearest-neighbors search problem, which is of fundamental importance in information retrieval, data compression, and machine learning (*10*). Each image is typically represented as a d-dimensional vector of feature values. (Each odor that a fly processes is a 50-dimensional feature vector of firing rates.) A distance metric is used to compute the similarity between two images (feature vectors), and the goal is to efficiently find the nearest neighbors of any query image. If the web contained only a few images, then brute force linear search could easily be used to find the exact nearest neighbors. If the web contained many images, but each image was represented by a low-dimensional vector (e.g., 10 or 20 features), then space-partitioning methods (*12*) would similarly suffice. However, for large databases with high-dimensional data, neither approach scales (*11*).

In many applications, returning an approximate set of nearest neighbors that are "close enough" to the query is adequate, so long as they can be found quickly. This has motivated an approach for finding approximate nearest neighbors by LSH (*10*). For the fly, as noted, the locality-sensitive property states that two odors that generate similar ORN responses will be represented by two tags that are themselves similar (Fig. 1B). Likewise, for image search, the tag of an elephant image will be more similar to the tag of another elephant image than to the tag of a skyscraper image.

Unlike a traditional (non-LSH) hash function, where the input points are scattered randomly and uniformly over the range, a LSH function provides a distance-preserving embedding of points from d-dimensional space into m-dimensional space (the latter corresponds to the tag). Thus, points that are closer to one another in input space have a higher probability of being assigned the same or a similar tag than points that are far apart. [A formal definition is given in (*13*).]

To design a LSH function, one common trick is to compute random projections of the input data (*10*, *11*)—that is, to multiply the input feature vector by a random matrix. The Johnson-Lindenstrauss lemma (*14*, *15*) and its many variants (*16*–*18*) provide strong theoretical bounds on how well locality is preserved when embedding data from d into m dimensions by using various types of random projections.

The fly also assigns tags to odors through random projections (step 2 in Fig. 1A; 50 PNs → 2000 KCs), which provides a key clue to the function of this part of the circuit. There are, however, three differences between the fly algorithm and conventional LSH algorithms. First, the fly uses sparse, binary random projections, whereas LSH functions typically use dense, Gaussian random projections that require many more mathematical operations to compute. Second, the fly expands the dimensionality of the input after projection ($d \ll m$), whereas LSH reduces the dimensionality ($d \gg m$). Third, the fly sparsifies the higher-dimensionality representation by a WTA mechanism, whereas LSH preserves a dense representation.

[1]Department of Computer Science and Engineering, University of California San Diego, La Jolla, CA, USA. [2]Molecular Neurobiology Laboratory, The Salk Institute for Biological Studies, La Jolla, CA, USA. [3]Kavli Institute for Brain and Mind, University of California San Diego, La Jolla, CA, USA. [4]Integrative Biology Laboratory, The Salk Institute for Biological Studies, La Jolla, CA, USA.
*Corresponding author. Email: navlakha@salk.edu

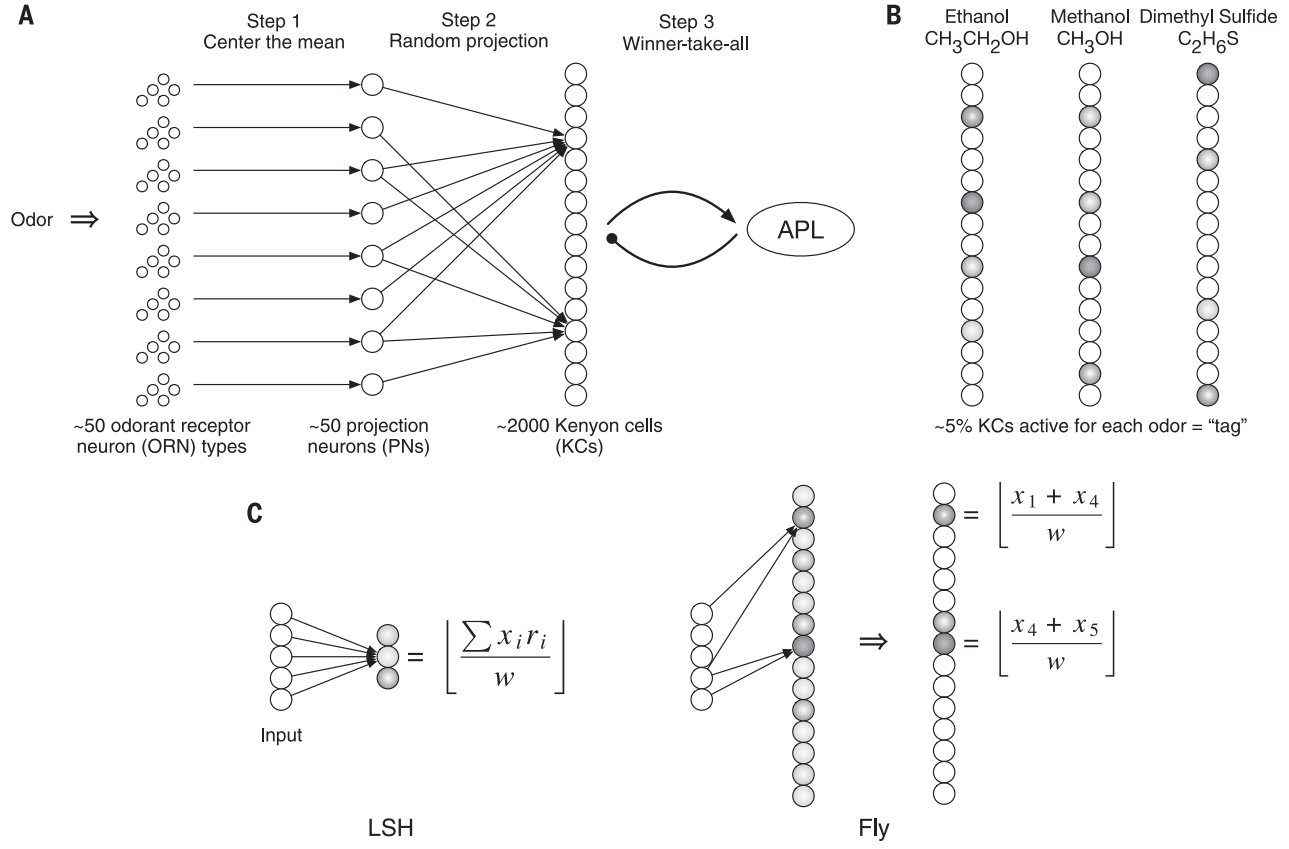

Fig. 1. Mapping between the fly olfactory circuit and locality-sensitive hashing (LSH). (**A**) Schematic of the fly olfactory circuit. In step 1, 50 ORNs in the fly's nose send axons to 50 PNs in the glomeruli; as a result of this projection, each odor is represented by an exponential distribution of firing rates, with the same mean for all odors and all odor concentrations. In step 2, the PNs expand the dimensionality, projecting to 2000 KCs connected by a sparse, binary random projection matrix. In step 3, the KCs receive feedback inhibition from the anterior paired lateral (APL) neuron, which leaves only the top 5% of KCs to remain firing spikes for the odor. This 5% corresponds to the tag (hash) for the odor. (**B**) Illustrative odor responses. Similar pairs of odors (e.g., methanol and ethanol) are assigned more similar tags than are dissimilar odors. Darker shading denotes higher activity. (**C**) Differences between conventional LSH and the fly algorithm. In the example, the computational complexity for LSH and the fly are the same. The input dimensionality $d = 5$. LSH computes $m = 3$ random projections, each of which requires 10 operations (five multiplications plus five additions). The fly computes $m = 15$ random projections, each of which requires two addition operations. Thus, both require 30 total operations. **x**, input feature vector; r, Gaussian random variable; w, bin width constant for discretization.

In the supplementary materials (*13*), we show analytically that sparse, binary random projections of the type in the fly olfactory circuit generate tags that preserve the neighborhood structure of input points. This proves that the fly's circuit represents a previously unknown LSH family.

We then empirically evaluated the fly algorithm versus traditional LSH (*10*, *11*) on the basis of how precisely each algorithm could identify nearest neighbors of a given query point. To perform a fair comparison, we fixed the computational complexity of both algorithms to be the same (Fig. 1C). That is, the two approaches were fixed to use the same number of mathematical operations to generate a hash of length k (i.e., a vector with k nonzero values) for each input (*13*).

We compared the two algorithms for finding nearest neighbors in three benchmark data sets: SIFT ($d = 128$), GLOVE ($d = 300$), and MNIST ($d = 784$) (*13*). SIFT and MNIST both contain vector representations of images used for image similarity search, whereas GLOVE contains vector representations of words used for semantic similarity search. We used a subset of each data set with 10,000 inputs each, in which each input was represented as a feature vector in d-dimensional space. To test performance, we selected 1000 random query inputs from the 10,000 and compared true versus predicted nearest neighbors. That is, for each query, we found the top 2% (200) of its true nearest neighbors in input space, determined on the basis of Euclidean distance between feature vectors. We then found the top 2% of predicted nearest neighbors in m-dimensional hash space, determined on the basis of the Euclidean distance between tags (hashes). We varied the length of the hash (k) and computed the overlap between the ranked lists of true and predicted nearest neighbors by using the mean average precision (*19*). We averaged the mean average precision over 50 trials, in which, for each trial, the random projection matrices and the queries changed. We isolated each of the three differences between the fly algorithm and LSH to test their individual effect on nearest-neighbors retrieval performance.

Replacing the dense Gaussian random projection of LSH with a sparse binary random projection did not hurt how precisely nearest neighbors could be identified (Fig. 2A). These results support our theoretical calculations that the fly's random projection is locality-sensitive. Moreover, the sparse, binary random projection achieved a computational savings of a factor of 20 relative to the dense, Gaussian random projection (fig. S1) (*13*).

When expanding the dimensionality, sparsifying the tag using WTA resulted in better performance than using random tag selection (Fig. 2B). WTA selected the top k firing KCs as the tag, unlike random tag selection, which selected k random KCs. For both, we used $20k$ random projections for the fly to equate the number of mathematical operations used by the fly and LSH (*13*). For example, for the SIFT data set with

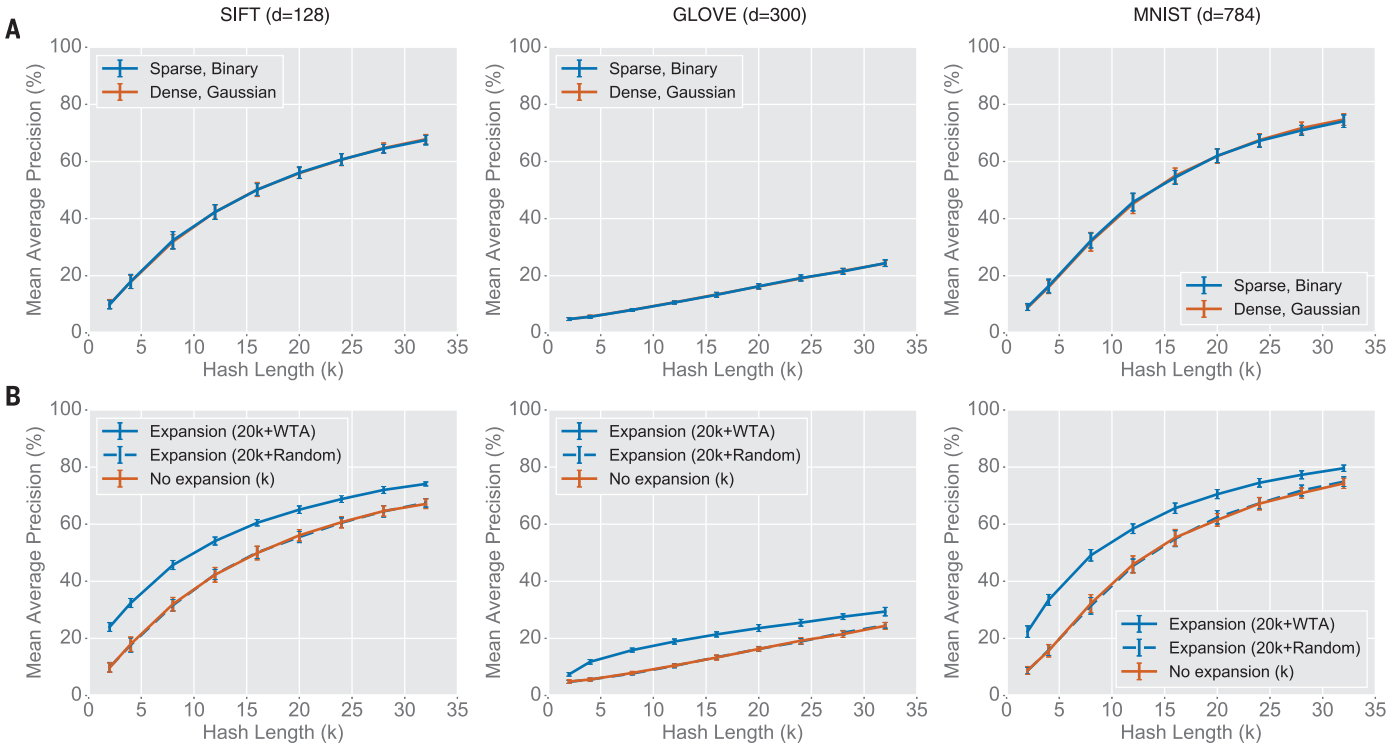

Fig. 2. Empirical comparison of different random projection types and tag-selection methods. In all plots, the *x* axis is the length of the hash, and the *y* axis is the mean average precision denoting how accurately the true nearest neighbors are found (higher is better). (**A**) Sparse, binary random projections offer near-identical performance to that of dense, Gaussian random projections, but the former provide a large savings in computation. (**B**) The expanded-dimension (from *k* to 20*k*) plus winner-take-all (WTA) sparsification further boosts performance relative to non-expansion. Results are consistent across all three benchmark data sets. Error bars indicate standard deviation over 50 trials.

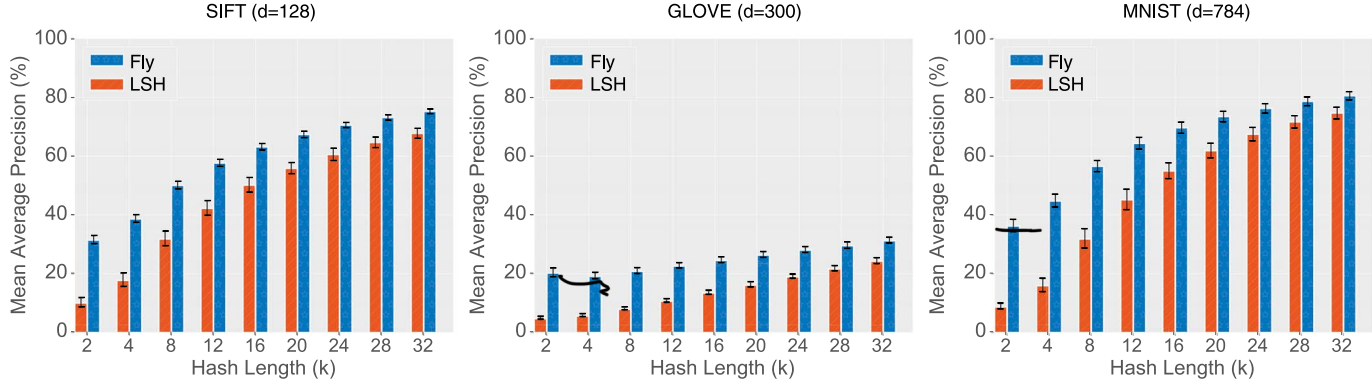

Fig. 3. Overall comparison between the fly algorithm and LSH. In all plots, the *x* axis is the length of the hash, and the *y* axis is the mean average precision (higher is better). A 10*d* expansion was used for the fly. Across all three data sets, the fly's method outperforms LSH, most prominently for short hash lengths. Error bars indicate standard deviation over 50 trials.

hash length k = 4, random selection yielded a 17.7% mean average precision, versus roughly double that (32.4%) using WTA. Thus, selecting the top firing neurons best preserves relative distances between inputs; the increased dimensionality also makes it easier to segregate dissimilar inputs. For random tag selection, we selected k random (but fixed for all inputs) KCs for the tag; hence, its performance is effectively the same as doing k random projections, as in LSH.

With further expansion of the dimensionality (from 20k to 10d KCs, closer to the actual fly's circuitry), we obtained further gains relative to LSH in identifying nearest neighbors across all data sets and hash lengths (Fig. 3). The gains were highest for very short hash lengths, where there was an almost threefold improvement in mean average precision (e.g., for MNIST with k = 4, 16.0% for LSH, versus 44.8% for the fly algorithm).

We also found similar gains in performance when testing the fly algorithm in higher dimensions and for binary LSH (*20*) (figs. S2 to S3). Thus, the fly algorithm is scalable and may be useful across other LSH families.

Our work identified a synergy between strategies for similarity matching in the brain (*21*) and hashing algorithms for nearest-neighbors search in large-scale information retrieval systems. It may also have applications in duplicate detection, clustering, and energy-efficient deep learning (*22*). There are numerous extensions to LSH (*23*), including the use of multiple hash tables (*11*) to boost precision (we used one for both algorithms), the use of multiprobe (*24*) so that similar tags can be grouped together (which may be easier to implement for the fly algorithm

Table 1. The generality of locality-sensitive hashing in the brain. Shown are the steps used in the fly olfactory circuit and their potential analogs in vertebrate brain regions.

	Step 1	Random projection	Step 2 (expansion)	Step 3 (WTA)
Fly olfaction	Antennae lobe; 50 glomeruli	Sparse, binary; samples six glomeruli	Mushroom body; 2000 Kenyon cells	APL neuron; top 5%
Mouse olfaction	Olfactory bulb; 1000 glomeruli	Dense, weak; samples all glomeruli	Piriform cortex; 100,000 semi-lunar cells	Layer 2A; top 10%
Rat cerebellum	Precerebellar nuclei	Sparse, binary; samples four precerebellar nuceli	Granule cell layer; 250 million granule cells	Golgi cells; top 10 to 20%
Rat hippocampus	Entorhinal cortex; 30,000 grid cells	Unknown	Dentate gyrus; 1.2 million granule cells	Hilar cells; top 2%

because tags are sparse), various quantization tricks for discretizing hashes (*25*), and learning [called data-dependent hashing (*13*)]. There are also methods to speed up the random projection multiplication, both for LSH schemes by fast Johnson-Lindenstrauss transforms (*26*, *27*) and for the fly by fast sparse matrix multiplication. Our goal was to fairly compare two conceptually different approaches for the nearest-neighbors search problem; in practical applications, all of these extensions will need to be ported to the fly algorithm.

Some of the fly algorithm's strategies have been used before. For example, MinHash (*28*) and winner-take-all hash (*29*) both use WTA-like components, though neither propose expanding the dimensionality; similarly, random projections are used in many LSH families, but none, to our knowledge, use sparse, binary projections. The fly olfactory circuit appears to have evolved to use a distinctive combination of these computational ingredients. The three hallmarks of the fly's circuit motif may also appear in other brain regions and species (Table 1). Thus, locality-sensitive hashing may be a general principle of computation used in the brain (*30*).

REFERENCES AND NOTES

1. C. F. Stevens, *Proc. Natl. Acad. Sci. U.S.A.* **112**, 9460–9465 (2015).
2. D. Owald, S. Waddell, *Curr. Opin. Neurobiol.* **35**, 178–184 (2015).
3. G. C. Turner, M. Bazhenov, G. Laurent, *J. Neurophysiol.* **99**, 734–746 (2008).
4. A. C. Lin, A. M. Bygrave, A. de Calignon, T. Lee, G. Miesenböck, *Nat. Neurosci.* **17**, 559–568 (2014).
5. M. Papadopoulou, S. Cassenaer, T. Nowotny, G. Laurent, *Science* **332**, 721–725 (2011).
6. E. A. Hallem, J. R. Carlson, *Cell* **125**, 143–160 (2006).
7. C. F. Stevens, *Proc. Natl. Acad. Sci. U.S.A.* **113**, 6737–6742 (2016).
8. S. R. Olsen, V. Bhandawat, R. I. Wilson, *Neuron* **66**, 287–299 (2010).
9. S. J. Caron, V. Ruta, L. F. Abbott, R. Axel, *Nature* **497**, 113–117 (2013).
10. A. Andoni, P. Indyk, *Commun. ACM* **51**, 117 (2008).
11. A. Gionis, P. Indyk, R. Motwani, in *VLDB'99, Proceedings of the 25th International Conference on Very Large Data Bases*, M. P. Atkinson *et al*., Eds. (Morgan Kaufman, 1999), pp. 158–529.
12. H. Samet, *Foundations of Multidimensional and Metric Data Structures* (Morgan Kaufmann Series in Computer Graphics and Geometric Modeling, Morgan Kaufmann, 2005).
13. Materials and methods are available as supplementary materials.
14. W. Johnson, J. Lindenstrauss, in *Conference on Modern Analysis and Probability*, R. Beals, A. Beck, A. Bellow, A. Hajian, Eds., vol. 26 of *Contemporary Mathematics* (American Mathematical Society, 1984), pp. 189–206.
15. S. Dasgupta, A. Gupta, *Random Structures Algorithms* **22**, 60–65 (2003).
16. D. Achlioptas, *J. Comput. Syst. Sci.* **66**, 671–687 (2003).
17. Z. Allen-Zhu, R. Gelashvili, S. Micali, N. Shavit, *Proc. Natl. Acad. Sci. U.S.A.* **111**, 16872–16876 (2014).
18. D. Kane, J. Nelson, *J. Assoc. Comput. Mach.* **61**, 4 (2014).
19. Y. Lin, R. Jin, D. Cai, S. Yan, X. Li, in *2013 IEEE Conference on Computer Vision and Pattern Recognition* (IEEE Computer Society, 2013), pp. 446–451.
20. M. S. Charikar, in *Proceedings of the Thirty-Fourth Annual ACM Symposium on Theory of Computing*, *STOC '02* [Association for Computing Machinery (ACM), 2002], pp. 380–388.
21. C. Pehlevan, D. B. Chklovskii, in *NIPS'15, Proceedings of the 28th International Conference on Neural Information Processing Systems* (MIT Press, 2015), pp. 2269–2277.
22. R. Spring, A. Shrivastava, Scalable and sustainable deep learning via randomized hashing. arXiv:1602.08194 [stat.ML] (26 February 2016).
23. M. Slaney, Y. Lifshits, J. He, *Proc. IEEE* **100**, 2604–2623 (2012).
24. Q. Lv, W. Josephson, Z. Wang, M. Charikar, K. Li, in *VLDB '07, Proceedings of the 33rd International Conference on Very Large Data Bases* (ACM, 2007), pp. 950–961.
25. P. Li, M. Mitzenmacher, A. Shrivastava, in *Proceedings of the 31st International Conference on Machine Learning* (Proceedings of Machine Learning Research, 2014), pp. 676–684.
26. A. Dasgupta, R. Kumar, T. Sarlos, in *KDD '11, The 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (ACM, 2011), pp. 1073–1081.
27. A. Andoni, P. Indyk, T. Laarhoven, I. Razenshteyn, L. Schmidt, in *NIPS'15, Proceedings of the 28th International Conference on Neural Information Processing Systems* (MIT Press, 2015), pp. 1225–1233.
28. A. Broder, in *Proceedings of the Compression and Complexity of Sequences 1997* (IEEE Computer Society, 1997), p. 21.
29. J. Yagnik, D. Strelow, D. A. Ross, R.-s. Lin, in *2011 International Conference on Computer Vision* (IEEE Computer Society, 2011), pp. 2431–2438.
30. L. G. Valiant, *Curr. Opin. Neurobiol.* **25**, 15–19 (2014).

ACKNOWLEDGMENTS

For funding support, C.F.S. thanks the NSF (grant EAGER PHY-1444273), and S.N. thanks the Army Research Office (grant DOD W911NF-17-1-0045). All authors thank A. Lang and J. Berkowitz for helpful comments on the manuscript. Code and data are available at https://bitbucket.org/navlakha/flylsh.

SUPPLEMENTARY MATERIALS

www.sciencemag.org/content/358/6364/793/suppl/DC1
Materials and Methods
Supplementary Text
Figs. S1 to S3
References (*31*–*49*)

14 February 2017; accepted 25 September 2017
10.1126/science.aam9868

Science

A neural algorithm for a fundamental computing problem

Sanjoy Dasgupta, Charles F. Stevens and Saket Navlakha

Science **358** (6364), 793-796.
DOI: 10.1126/science.aam9868

Fly brain inspires computing algorithm

Flies use an algorithmic neuronal strategy to sense and categorize odors. Dasgupta *et al.* applied insights from the fly system to come up with a solution to a computer science problem. On the basis of the algorithm that flies use to tag an odor and categorize similar ones, the authors generated a new solution to the nearest-neighbor search problem that underlies tasks such as searching for similar images on the web.

Science, this issue p. 793