PLOS COMPUTATIONAL BIOLOGY

METHODS

Connectivity Matrix Seriation via Relaxation

Alexander Borst *

Max-Planck-Institute for Biological Intelligence, Martinsried, Germany

* alexander.borst@bi.mpg.de

Abstract

Volume electron microscopy together with computer-based image analysis are yielding neural circuit diagrams of ever larger regions of the brain. These datasets are usually represented in a cell-to-cell connectivity matrix and contain important information about prevalent circuit motifs allowing to directly test various theories on the computation in that brain structure. Of particular interest are the detection of cell assemblies and the quantification of feedback, which can profoundly change circuit properties. While the ordering of cells along the rows and columns doesn't change the connectivity, it can make special connectivity patterns recognizable. For example, ordering the cells along the flow of information, feedback and feedforward connections are segregated above and below the main matrix diagonal, respectively. Different algorithms are used to renumber matrices such as to minimize a given cost function, but either their performance becomes unsatisfying at a given size of the circuit or the CPU time needed to compute them scales in an unfavorable way with increasing number of neurons. Based on previous ideas, I describe an algorithm which is effective in matrix reordering with respect to both its performance as well as to its scaling in computing time. Rather than trying to reorder the matrix in discrete steps, the algorithm transiently relaxes the integer program by assigning a real-valued parameter to each cell describing its location on a continuous axis ('smooth-index') and finds the parameter set that minimizes the cost. I find that the smooth-index algorithm outperforms all algorithms I compared it to, including those based on topological sorting.

Author summary

Connectomic data provide researchers with neural circuit diagrams of ever larger regions of the brain. These datasets are usually represented in a cell-to-cell connectivity matrix and contain important information about prevalent circuit motifs. Such motifs, however, only become visible if the connectivity matrix is reordered appropriately. For example, ordering the cells along the flow of information, feedback and feedforward connections are segregated above and below the main matrix diagonal, respectively. While most previous approaches rely on topological sorting, my method relaxes the discrete vertex indices to real numbers ('smooth-index') along independent parameter axes and defines a differentiable cost function, thus, allowing gradient-based algorithms to find a minimum. The parameter set at this minimum is then re-discretized to reorder the connectivity matrix

Citation: Borst A (2024) Connectivity Matrix Seriation via Relaxation. PLoS Comput Biol 20(2): e1011904. https://doi.org/10.1371/journal.pcbi.1011904

Editor: Matthieu Louis, University of California Santa Barbara, UNITED STATES

Received: August 24, 2023

Accepted: February 9, 2024

Published: February 20, 2024

Data Availability Statement: All code and data is available at: https://github.com/axelborst/Matrix_Reordering.

Funding: AB: Funding was obtained from my general budget of the Max-Planck-Society. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Competing interests: No competing interests.

accordingly. I find my method to scale favorably with the circuit size and to outperform all algorithms I compared it to.

Introduction

Connectomic studies provide neural circuit diagrams of ever larger regions of the brain [1–11]. A natural way to order the elements of such circuit diagrams is along the direction of information flow. This way, import features of the circuit become immediately visible in the respective connectivity matrix: synfire chains [12,13] will form diagonals of the matrix, and squared blocks along the diagonal will indicate the presence of cell assemblies. Arranging the outputs of each neuron within the columns of the connectivity matrix, the ordering will put forward synapses in the lower and recurrent connections in the upper triangle. Isolating recurrent synapses can be achieved by ordering the connectivity matrix such as to push as many entries to the lower triangle as possible, a so-called 'minimum feedback arc set'. I focus on recurrency as a specific property of neural circuits because of the importance recurrent synapses have for the temporal processing properties of the circuit. In networks without feed-back, the eigenvalues of the corresponding dynamical system matrix $A = T^{-1}(M-I)$—with T being the matrix holding the cellular time-constants along the diagonal, M being the connectivity matrix and I the identity matrix—are the negative inverse of the cellular time-constants [14]. In contrast, in networks with feed-back, the eigenvalues depend not only on the cellular time-constants, but in addition on the connectivity parameters [15–17]. Thus, feed-back generally generates new time constants and thereby allows for signal processing at time scales that can go beyond the ones of the isolated elements by orders of magnitude, thereby providing a substrate for working memory [18–20], path integration [21,22] or delay lines in the context of motion vision [23,24]. Within a given connectomic data set, however, neurons usually are not numbered according to the main flow of information within the circuit. In order to retrieve information from the dataset, neurons must therefore be reordered with a specific objective in mind. Reordering of a square matrix with size N x N starts with an index list $\pi = (\pi_1, \pi_2 \ldots \pi_N)$ which assigns each circuit element i the new index π_i. From this, a permutation matrix P is created by permuting the columns of the identity matrix accordingly, and the reordered connectivity matrix R is obtained by $R = PMP^{-1}$. Since renumbering preserves the connectivity information, the graphs resulting from renumbering are all isomorphic. Different algorithms exist to reorder matrices [25–33, for review, see 17,34,35]. As a fundamental common property, they all retain the discrete nature of the vertex indices. This way, any property defined to be optimized is not a differentiable function along any parameter axes, but rather a discrete value depending on a given permutation. Since a full enumeration of all permutations grows with O(N!), a brute force approach fails quickly as N increases. The problem presents itself as a sorting problem where each of the different algorithms apply a different, particular strategy.

Results

My approach is different from topological searches but rather based on integer relaxation techniques [36–40] (Fig 1). I first relax each vertex index from being a discrete integer number and turn it into a smooth real number z_i that indicates the respective vertex position on a continuous axis. Next, each vertex position is considered to be a parameter value along an independent axis. Thus, the cost to be minimized becomes a function of an N-dimensional parameter space. Although the number of possible solutions, i.e. all permutations of N, represent only a small subspace of the new search space N^N, this transition brings the advantage that the cost

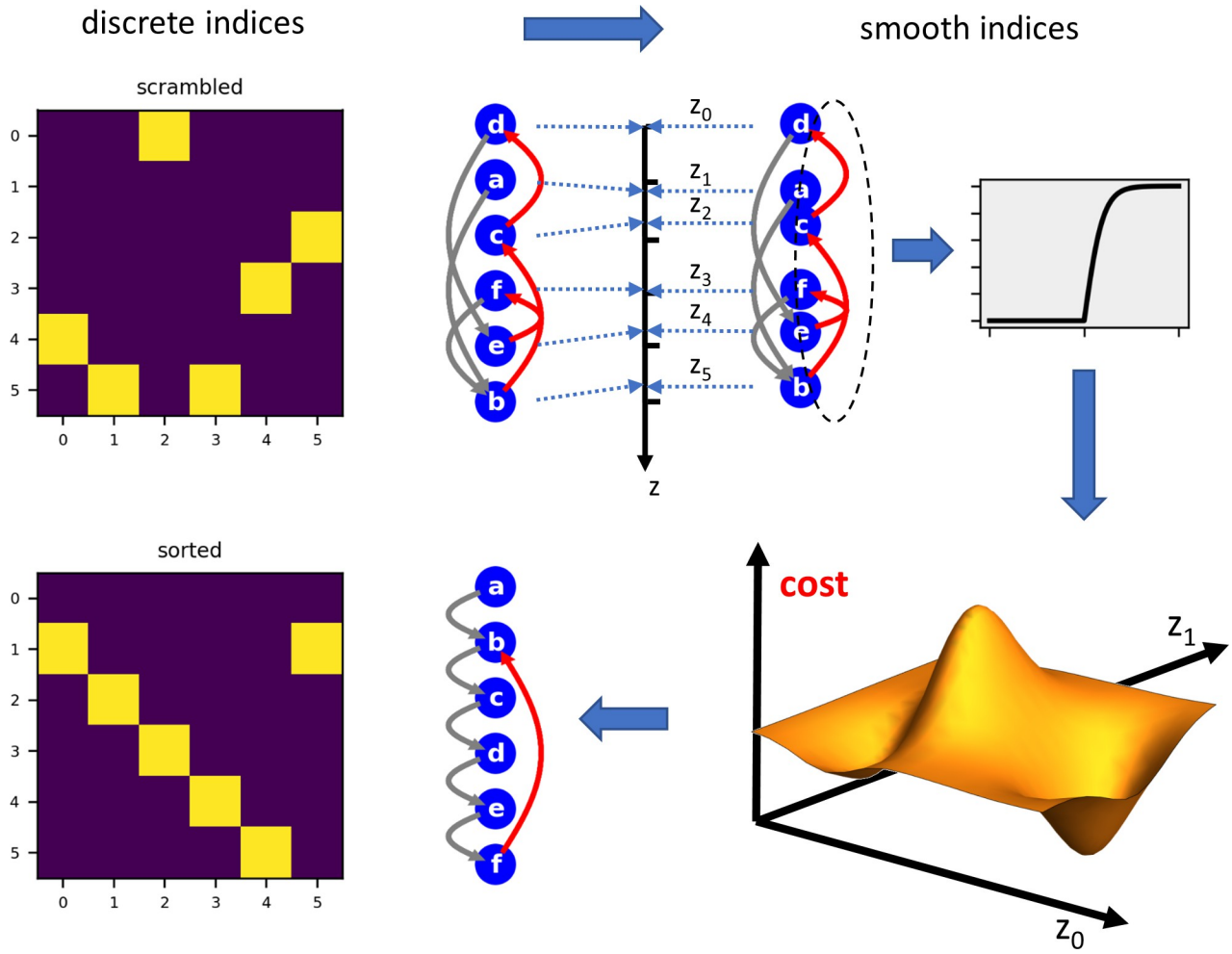

Fig 1. Working scheme of the smooth-index algorithm. Starting from a 'scrambled' circuit, the algorithm treats the indices as smooth values along independent parameter axes and minimizes a given cost function, in this case all recurrent connections.

https://doi.org/10.1371/journal.pcbi.1011904.g001

function can become differentiable, allowing gradient descent methods to be applied to search for a minimum. A recurrent connection is characterized by the fact that the position z_m of a presynaptic neuron m is larger than the position z_n of a postsynaptic neuron n. However, the number of recurrent connections varies in a discrete step at the point where $z_m = z_n$. Hence, it is not a differentiable function of z. I therefore chose the average length $z_m - z_n$ of all recurrent connections as a proxy for the number of recurrent connections, and, by taking a saturating function of the length, such as the logistic function, reduced the stronger influence of longer connections over shorter ones (Eq 1, Methods). To restrict the solutions to the permutation subspace, I require, as an additional criterion to the original cost function, each position to be different from all other positions. To this end, I define an additional 'Pauli term' as the mean of the squared differences between the positions and their rank, i.e. their place within the sorted array of positions (Eq 5, Methods). To return to discrete vertex indices, as the final step, the permutation list π that reorders the original matrix is obtained as the vector containing the arguments of the rank-sorted parameters at the minimum of the cost function.

I implemented the smooth-index algorithm in Python using the Scipy minimize function [41]. For a time-efficient search, I calculated the gradient or 'Jacobian' of the cost function (Eq 2 & Eq 6, Methods), which holds all first-order partial derivatives of the cost function along

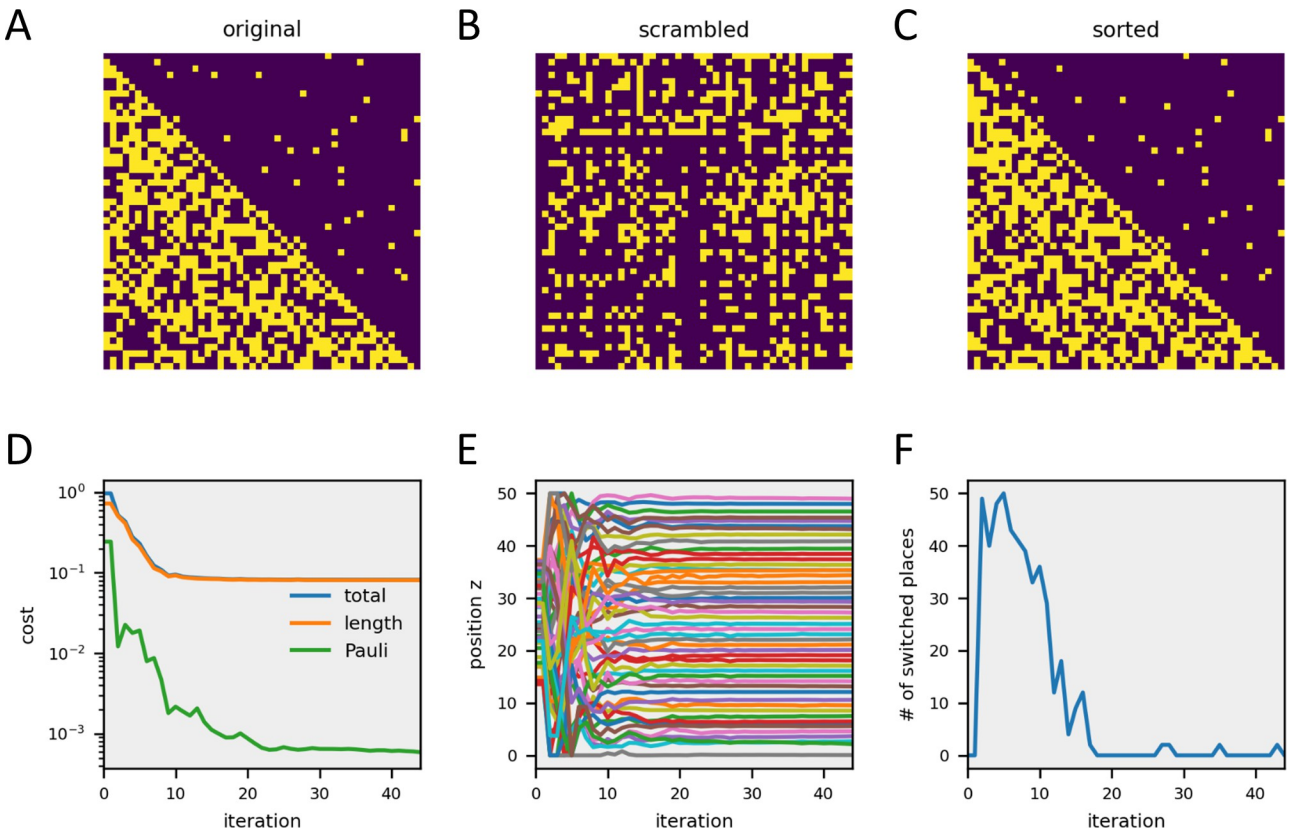

Fig 2. Example performance of the smooth-index algorithm. **A** Original matrix with a few recurrent connections. **B** Scrambled matrix, obtained by random index permutation. **C** Matrix resulting from smooth-index sorting. **D** Different cost functions as the algorithm performs the gradient descent. **E** Positions of all vertices z_i during gradient descent. **F** Number of switches, calculated as the number of differences between two consecutive rank-sorted position vectors, during gradient descent.

https://doi.org/10.1371/journal.pcbi.1011904.g002

each parameter z_n. The cost function as well as the Jacobian were passed on to the Scipy minimize function with randomly initialized parameter values z. As shown in Fig 2B–2G for a circuit with 50 neurons, the algorithm works well and isolates a similar number of recurrent synapses as is found in the original connectivity matrix.

To test the efficiency of the sorting algorithm quantitatively, I applied it to matrices of different sizes, from 10 up to 10 000 neurons. Each circuit was constructed randomly with a given probability for entries in the lower and upper triangle. To quantify the performance, I calculated the fraction of non-zero entries in the upper triangle of the original connectivity matrix, i.e. before scrambling, and compared that with the fraction after smooth-index sorting and after out-degree sorting. An example of such sorting for a circuit containing 50 neurons is shown in Fig 3A. For circuits with different sizes, the results demonstrate that the algorithm reorders the scrambled connectivity matrices with a high degree of fidelity, in particular for large numbers of neurons (Fig 3B, blue line). Topological sorting according to the out-degree of the nodes performs significantly worse (Fig 3B, red line). For the smooth-index algorithm, the CPU time needed to compute the reordered matrix on a standard desktop computer (Intel i9-7900X CPU with 10 cores, running at 3.3 GHz) grows from about 0.1 seconds for N = 10 to about 1000 seconds for N = 10 000. For larger networks, the CPU time roughly scales with O (N^2) (Fig 3C, blue line). Not astonishingly, out-degree sorting is always faster (Fig 3C, red line). The desktop was unable to handle networks with more than 20 000 neurons. All performance tests above were applied to matrices with a density of 50% in the lower triangle. In

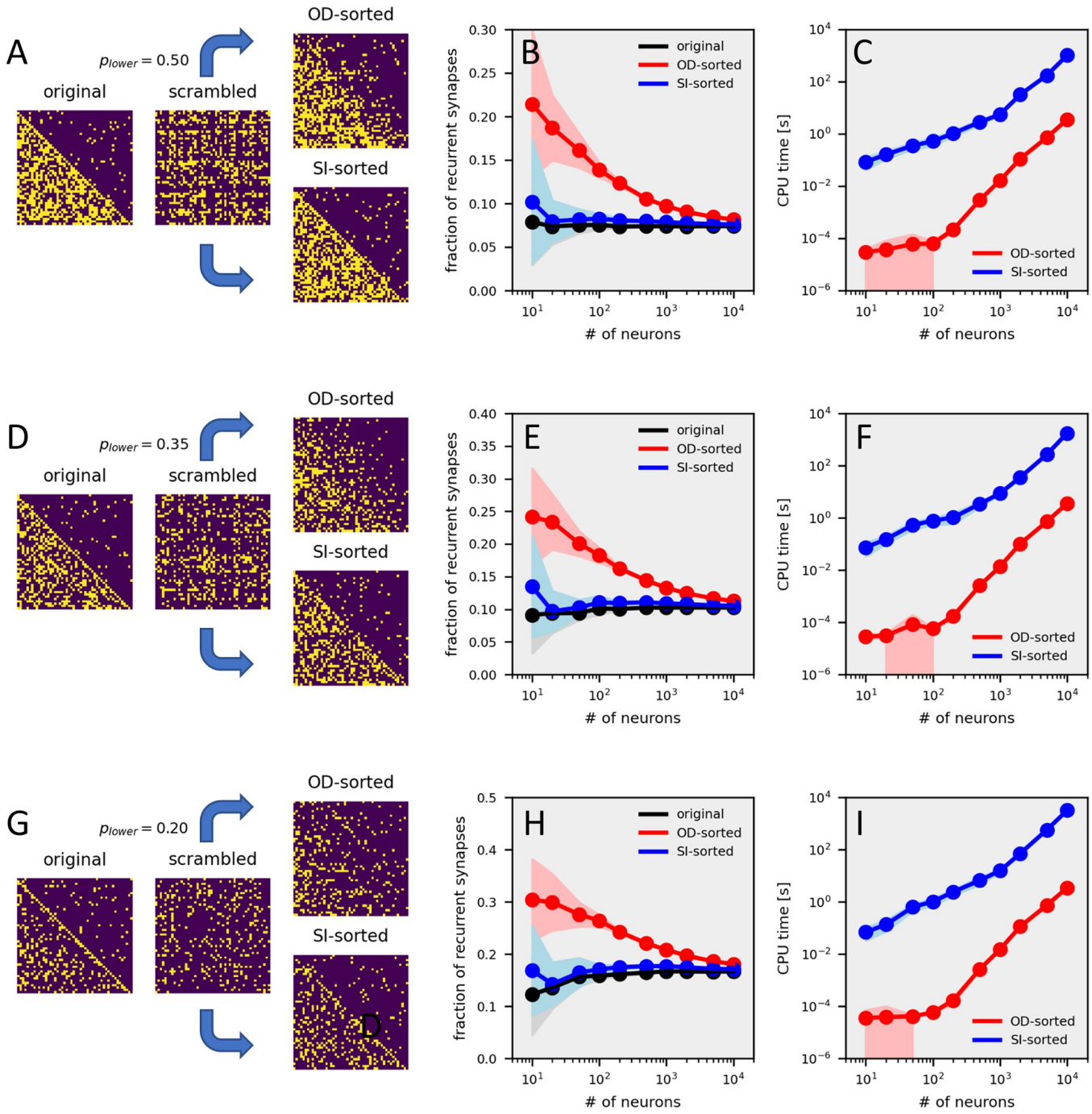

Fig 3. Performance of the smooth-index algorithm for matrices with different densities. A Example matrices for N = 50 neurons, given a density of 50% in the lower triangle**. B** Performance of the algorithm as a function of the number of neurons. Plotted is the fraction of non-zero entries in the upper triangle of the matrix in case of the original connectivity matrix before scrambling (in black), after out-degree sorting (in red) and after smooth-index sorting (in blue). **C** CPU time needed for out-degree sorting (in red) and for smooth-index sorting (in blue) as a function of the number of neurons. Data in B and C represent the mean +- standard deviation (shaded area) obtained from 10 sorting runs. **D-F** Same as above, but for a density of 35% in the lower triangle. **G-I** Same as above, but for a density of 20% in the lower triangle.

https://doi.org/10.1371/journal.pcbi.1011904.g003

order to see whether smooth-index sorting also performs well with lower density matrices, I ran the same procedure using matrices with a density of 35% (Fig 3D–3F) and 20% (Fig 3G–3I) density in the lower triangle. In all cases, the smooth-index algorithm yields better results than the out-degree sorting, reaching similar values as the original matrix, however with slightly increasing CPU time spent on matrices with lower densities.

I next compared smooth-index sorting with the algorithm developed to identify the minimum feedback arc set ('FAS'-algorithm) [28]. As a first test, I constructed an original connectivity matrix of 50 neurons with 50% density in the lower and 4% density in the upper triangle (Fig 4A) which was then scrambled randomly (Fig 4B). I then applied the FAS-algorithm to remove the minimum number of recurrent connections identified by the algorithm (Fig 4C). The difference of entries between this matrix and the original one allowed for determining the number of connections identified as being recurrent by the FAS algorithm. However, the remaining elements were still ordered as in the scrambled matrix. For a visual comparison with my algorithm, I then sorted the matrix using the Schur decomposition (Fig 4D). The decomposition uses a transformation matrix U (where $U^*U = I$, with I being the identity matrix), such that $M = U^*RU$, where R is now lower-triangular. If and only if the circuit is truly feed-forward, what should be the case when the FAS algorithm removes all recurrent connections, the transformation matrix U is a permutation matrix. I then applied the same permutation matrix U to sort the scrambled matrix with all its connections intact (Fig 4E) and compared the result with the matrix resulting from SI-sorting of the scrambled matrix (Fig 4F). I applied this procedure to original matrices with three different densities in the lower triangle, i.e. p = 50% (Fig 4G), p = 35% (Fig 4H) and p = 20% (Fig 4I), while the density in the upper triangle was held constant at 4%. Under all three conditions, the FAS algorithm overestimates the number of recurrent connections significantly (compare red and gray bars). In contrast, the SI algorithm results in similar values as found in the original matrix (compare blue and gray bars).

However, yielding a similar number of recurrent connections does not mean that these connections are identical to the ones in the original, unscrambled matrix. In order to see whether I can also identify recurrent synapses, I constructed a single connectivity matrix with about 12% recurrent synapses (Fig 5A), scrambled it once (Fig 5B) and subjected it to the smooth-index sorting 1000 times. With each run being randomly initialized, different minima were likely to be found each time. From all these different runs, I constructed a matrix which holds the probability values by which each connection was classified as recurrent (Fig 5C). Reversing the scrambling revealed that most recurrent synapses of the original matrix were classified as recurrent with a high probability (Fig 5D). Here, an especially interesting case are reciprocal connections between two cells. Given the fact that if one of the connections is feed-forward, the other must be recurrent, one might naively expect that, for reasons of symmetry, each of these reciprocal connections is classified as recurrent with a probability of 0.5. However, contrary to this expectation, looking at all pairs of reciprocal synapses, only one of the connections had a high, while the other had a low probability of being classified as recurrent (Fig 5E). Obviously, taking into account the full connectivity, the symmetry between reciprocally connected neurons breaks. In the end, my method correctly identified 82% of all recurrent connections of the original matrix (Fig 5F). I conclude that my method not only allows for calculating the degree of recurrency, i.e. how many recurrent connections exist in a given network, but also for successfully identifying these recurrent synapses and isolating them from the feed-forward ones.

I next applied my method to a connectome of 65 neurons within a single column of the optic lobe of the fruit fly *Drosophila melanogaster* [9]. This connectome is a weighted connectivity matrix where the non-zero entries indicate the number of synapses found for each

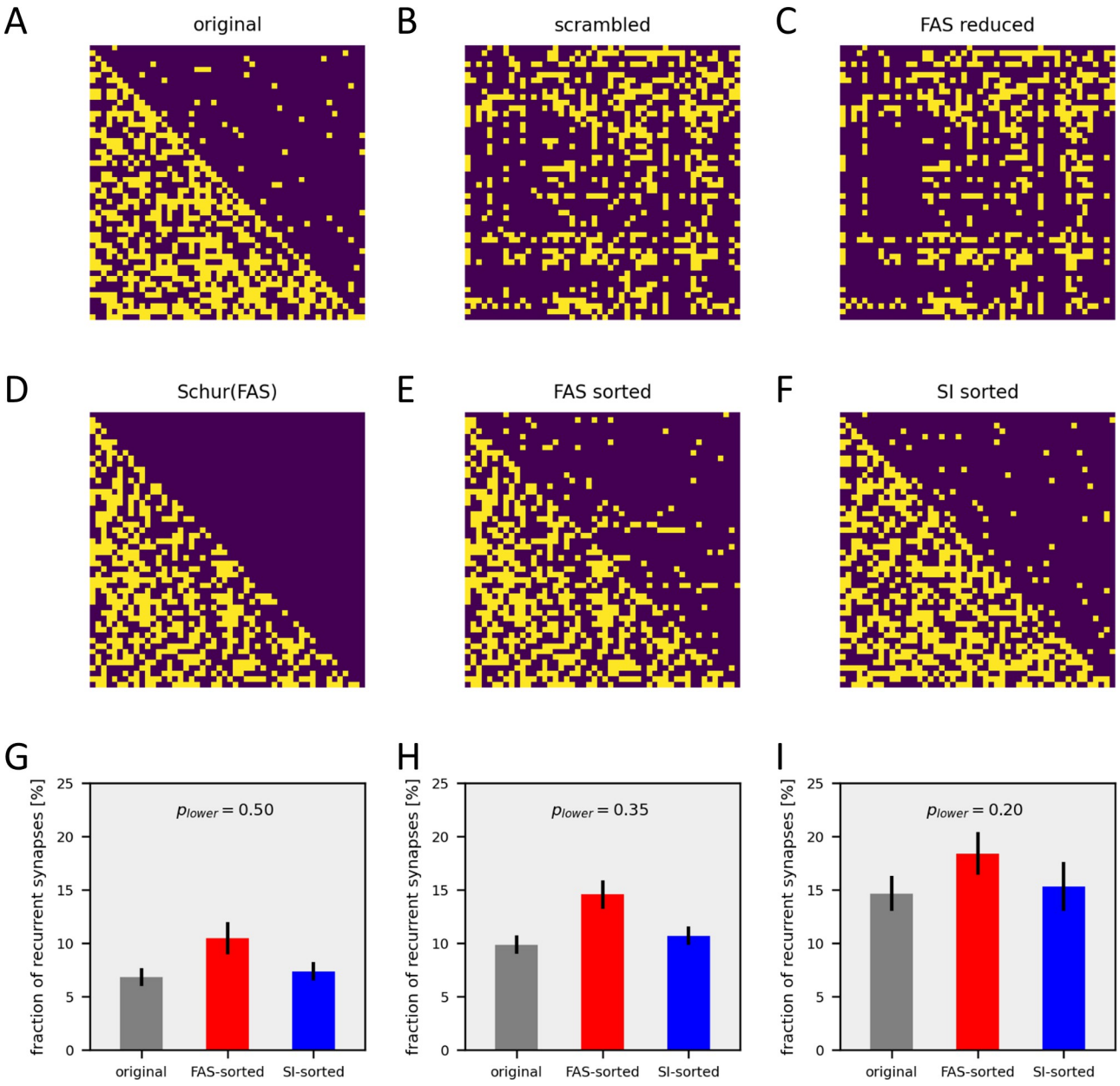

Fig 4. Comparison of the smooth-index ('SI') algorithm with feedback arc set ('FAS') sorting. A Original connectivity matrix of 50 neurons with 50% density in the lower and 4% density in the upper triangle. **B** Same as A, but after scrambling. **C** Resulting matrix after removing recurrent connections identified by the FAS algorithm. Note that the ordering of the scrambled matrix is retained. **D** FAS reduced matrix, reordered applying the Schur decomposition. **E** Full, scrambled matrix reordered applying the permutation matrix as obtained from applying the Schur decomposition to the FAS-reduced matrix. **F** Scrambled matrix after SI-sorting. Note that it has fewer entries in the upper triangle than the one in E. **G-I** Performance of the SI and the FAS algorithm for matrices with three different densities in the lower triangle, from p = 50% to p = 20%. The density in the upper triangle was held constant at 4%. Plotted is the fraction of non-zero entries in the upper triangle of the matrix in case of the original connectivity matrix before scrambling (in gray), after FAS sorting (in red) and after smooth-index sorting (in blue). Data represent the mean +- standard deviation obtained from 10 sorting runs.

https://doi.org/10.1371/journal.pcbi.1011904.g004

connection. Moreover, since neurons can be either excitatory or inhibitory, the sign of the connection is color coded such that excitatory connections are shown in red while inhibitory ones are shown in blue (Fig 6A). I reduced the weighted matrix into a 0,1-adjacency matrix by thresholding the absolute values of the weights at a level of four synapses (Fig 6B and 6C). This

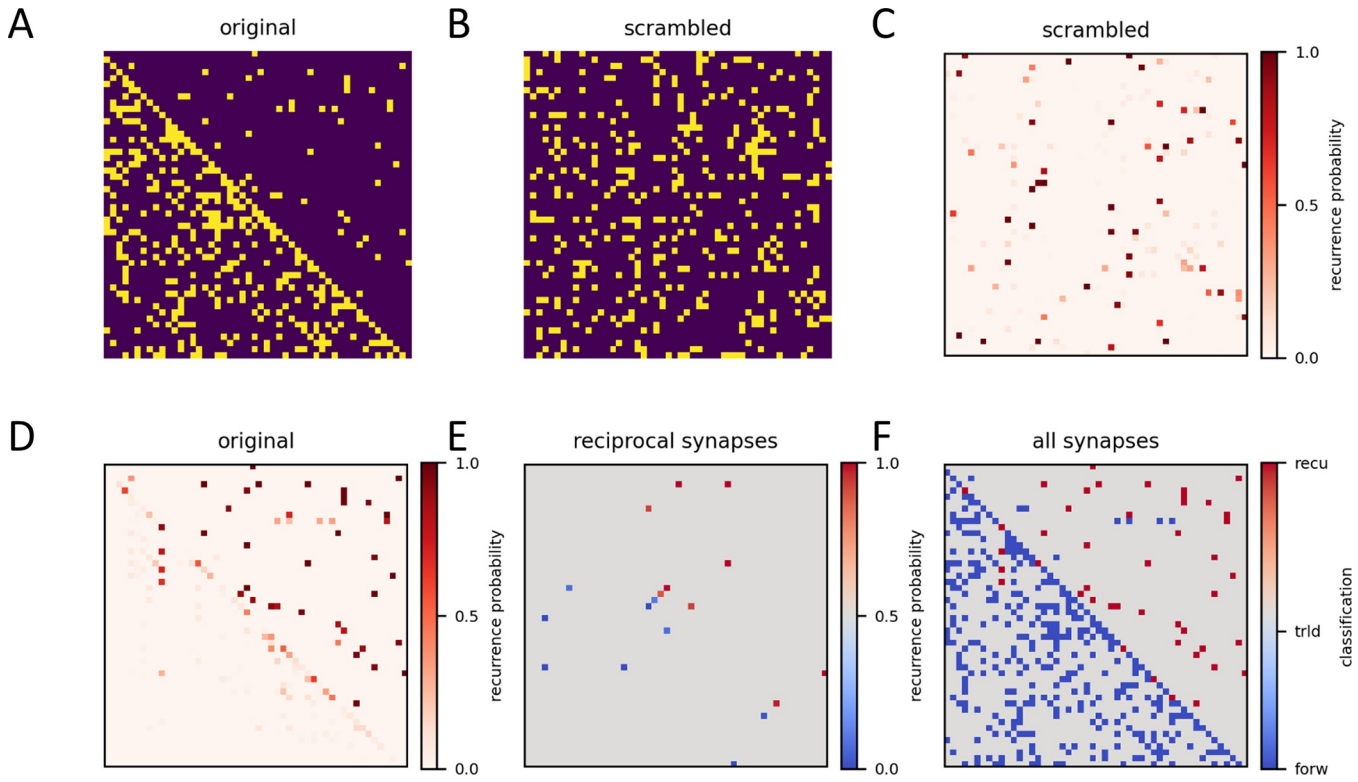

Fig 5. Identification of recurrent synapses by the smooth-index algorithm. A Original matrix. **B** Scrambled matrix. **C** Matrix holding the probability of each connection as being classified as recurrent from 1000 sorting runs. **D** Same as C, but after reversed scrambling. **E** Recurrence probability of all reciprocal connections. **F** Original connectivity matrix (in blue) with identified recurrent connections in red (probability threshold = 0.45).

https://doi.org/10.1371/journal.pcbi.1011904.g005

matrix was then reordered by the SI-algorithm 1000 times, which was again randomly initialized for each run. From all runs, the one with the lowest number of recurrent connections was chosen to reorder the original adjacency matrix, resulting in a reordered adjacency matrix (Fig 6D). The same reordering was then applied to the original, weighted connectivity matrix, resulting in a weighted, reordered matrix (Fig 6E). The original adjacency matrix had a total of 187 entries, with 67 entries in the upper triangle. 50 of the 187 connections were reciprocal. SI sorting resulted in a matrix with only 30 recurrent connections, i.e. 5 more than the absolute minimum, given 25 pairs of reciprocal connections. Of course, one doesn't know whether this is indeed the lower limit, but it is safe to state that at least 13% and at most 16% of the connections are recurrent.

In order to examine which connections were classified with a high probability, I determined, how often, in all 1000 runs, each connection was classified as recurrent. The result is shown in Table 1. Contrary to intuition, some connections between lamina neurons and medulla neurons are found to be recurrent with a high probability (row 2: L5 -> Mi1), although anatomy would suggest this to be feed-forward. In order to validate the above findings, I determined all simple paths from Mi1 to L5 and vice versa. Setting a maximum of 2 intermediate nodes, there are 20 paths from Mi1 to L5, but only 6 from L5 to Mi1, confirming the upstream placement of Mi1 with respect to L5. Furthermore, from reciprocally connected neuron pairs like Mi4 and Mi9, the connection from Mi4 to Mi9 is classified with a high probability as recurrent (row22: Mi4 -> Mi9). Again, as in Fig 5, taking into account the full connectivity breaks the symmetry between reciprocally connected neurons breaks defining one direction clearly as feed-forward and the other one as feedback. Five of the connections

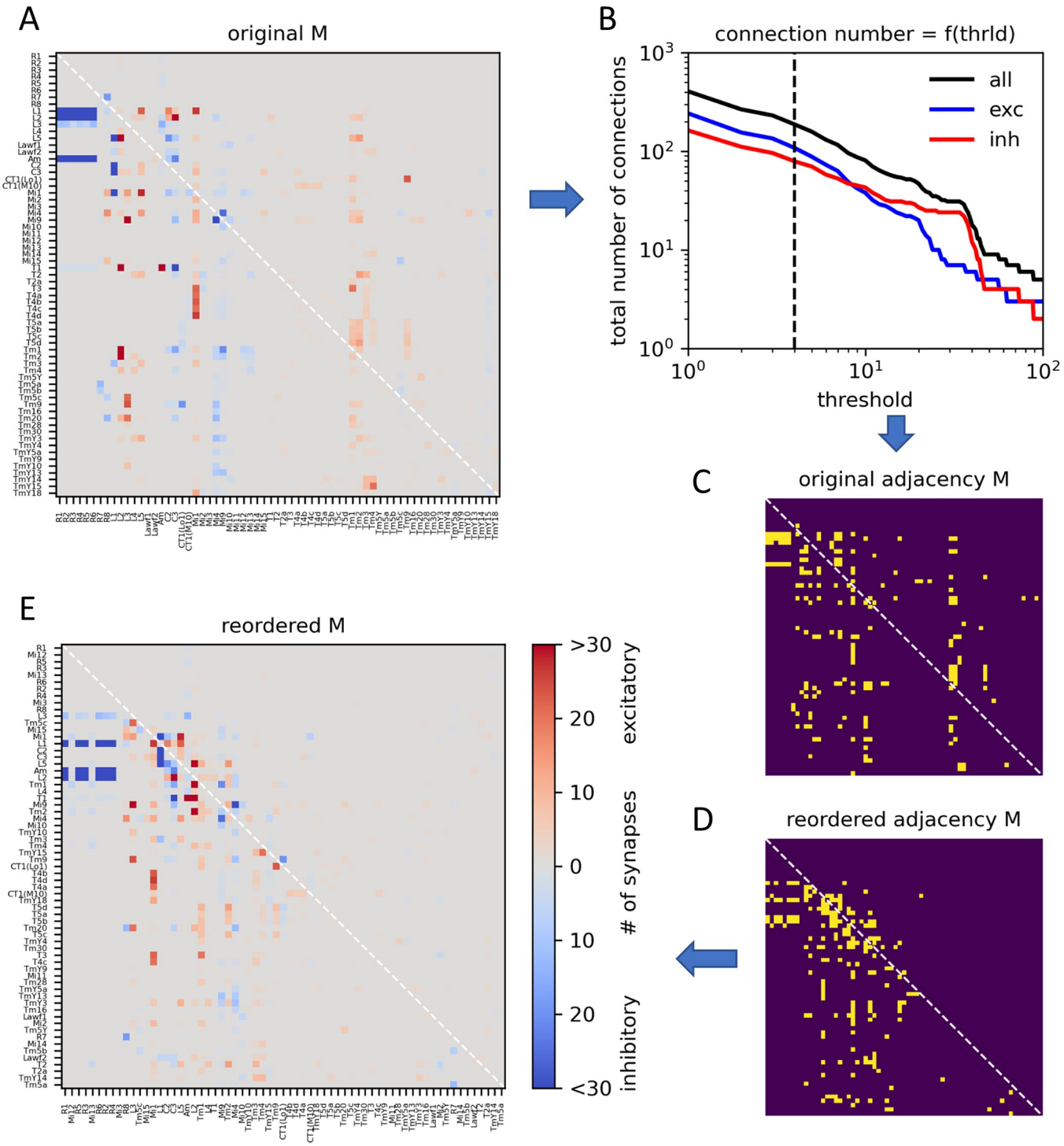

Fig 6. Application of the smooth-index algorithm to the single-column connectome of the Drosophila optic lobe. A Connectivity matrix with synaptic weights, ordered according to the sequence of neuropils, from retina to lamina to medulla. **B** Total number of connections as a function of threshold ('thrld') for all connections (in black) as well as for excitatory (in blue) and inhibitory (in red) connections, separately. **C** Resulting 0,1-adjacency matrix. **D** Reordered 0,1-adjacency matrix according to the least number of upper triangle entries obtained in 1000 runs of the smooth-index algorithm. **E** Same as D, but shown with the weights of the original weighted matrix.

https://doi.org/10.1371/journal.pcbi.1011904.g006

Table 1. List of recurrent connections from the single column connectome of Drosophila. Connections are sorted according to the probability by which a certain connection was classified (1st column, 'p_recurrency') as being recurrent in 1000 runs of the smooth-index algorithm. The values in the 2nd column ('Synaptic Strength') indicate the number of synapses at each connection, with positive values for excitatory, and inhibitory numbers for inhibitory connections. Highlighted are those connections which are not part of a reciprocally connected pair of neurons.

P_recurrency	Synapse strength	Pre	Post
1.000	4	Tm9	Tm2
0.999	28	L5	Mi1
0.995	4	L2	C3
0.994	8	L5	C2
0.993	-5	Mi4	Tm1
0.988	43	L2	L5
0.982	-19	CT1(Lo1)	Tm9
0.980	7	Tm2	Tm1
0.975	6	TmY10	Mi4
0.973	15	Tm2	L5
0.970	-5	Mi4	L2
0.970	10	Tm1	L5
0.968	-14	Am	L3
0.955	22	L5	L1
0.953	7	Tm2	L2
0.952	5	T1	L2
0.949	6	C3	L1
0.940	-136	L1	Mi1
0.933	-7	CT1(M10)	Mi1
0.833	-5	TmY15	Mi4
0.810	7	L5	C3
0.806	-42	Mi4	Mi9
0.775	5	Tm1	L2
0.763	10	Tm2	Mi9
0.755	-5	Mi9	Mi15
0.728	-20	Mi9	Tm1
0.717	-10	C2	Mi1
0.704	-41	L1	C2
0.698	-6	Mi10	Mi9
0.541	-5	Mi4	Tm2

https://doi.org/10.1371/journal.pcbi.1011904.t001

classified as recurrent are not from reciprocal pairs. These are Tm2 -> L5, Tm1 -> L5, Am -> L3, TmY15 -> Mi4 and Mi9 -> Mi15, highlighted in Table 1.

I finally explored whether the feedback detection problem is unique in being susceptible to smooth-index optimization. Another reordering problem with relevance to Neuroscience is the unmasking of clusters and processing chains. In order to apply the smooth-indexing method to identify properties like a block-diagonal structure or a limited band-width, I define the cost function as the mean squared length of all connections, irrespective of their sign, similar to an earlier approach [38] (Eq 3, Methods). Its gradient is given by Eq 4 (Methods). The algorithm successfully identifies the original structure of a band-width limited connectivity matrix (Fig 7A and 7B) as well as of a block-diagonal connectivity matrix (Fig 7C and 7D) with a high fidelity and outperforms the standard reverse Cuthill-McKee algorithm [25] over circuit sizes up to 10 000 neurons.

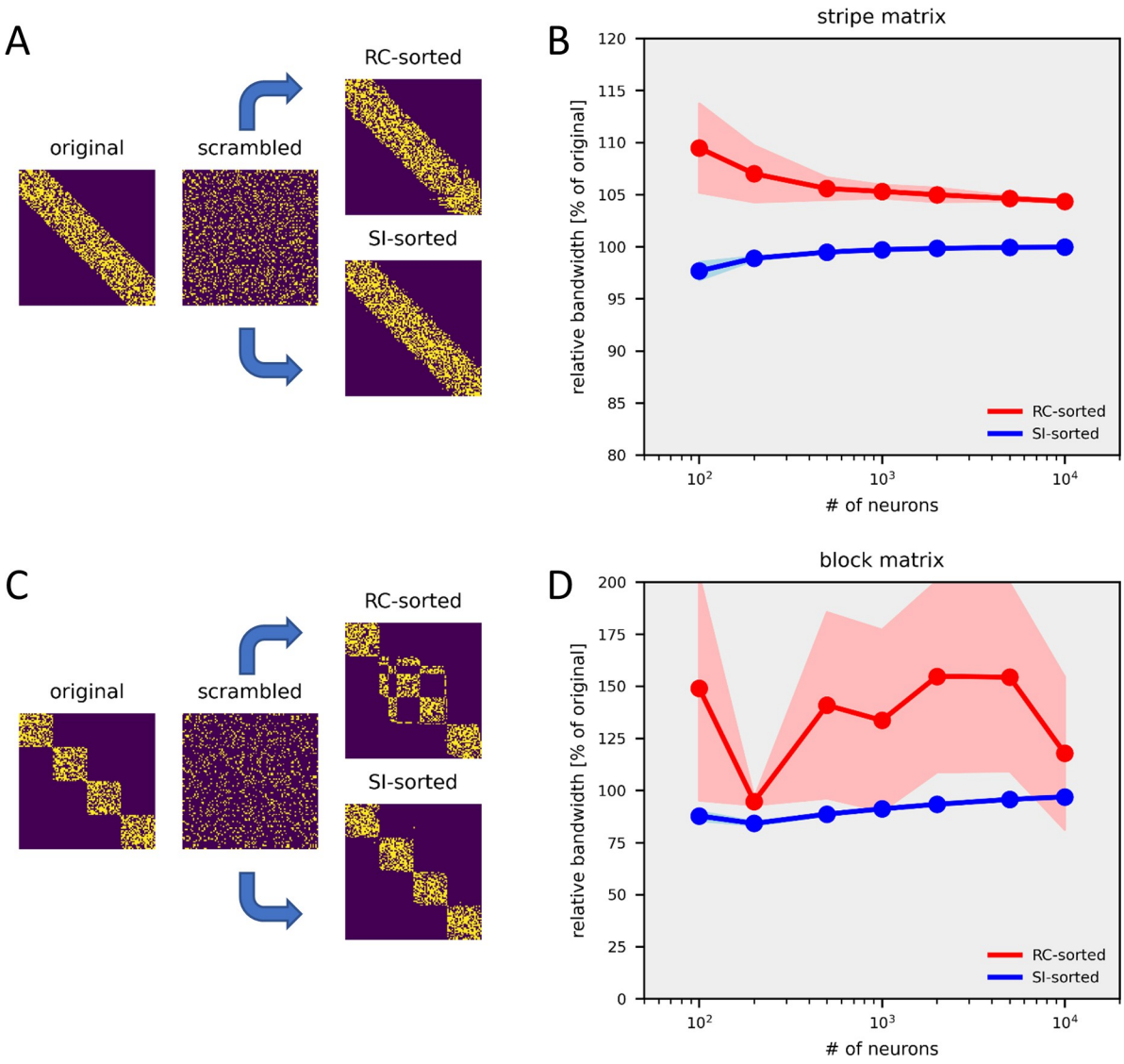

Fig 7. Application of the smooth-index to bandwidth-limited matrices (A,B) and to block-diagonal matrices (C,D). The smooth-index ('SI') algorithm is compared with the reverse Cuthill-McKee ('RC') algorithm. In A and C, one example sorting is shown for a 50 x 50 matrix. In B and D, the performance of both algorithms is quantified as the bandwidth of the reordered matrix relative to the original matrix. The performance is calculated as the average obtained from 10 runs +- the standard deviation (shaded area).

https://doi.org/10.1371/journal.pcbi.1011904.g007

Discussion

Searching for an optimum as a function of permutations within a given range is a hard problem because the number of possible permutations grows with N-factorial. The problem belongs to the large class of problems known as 'integer programs' or 'combinatorial optimization' which in general are NP-complete [42]. Therefore, various heuristic approaches have been taken such as the FAS algorithm [28–32], the Cuthill-McKee algorithm [25] or page-rank algorithms [33] (for review, see [17,34,35]). Depending on the particular objective and the specific algorithm, they are based on reordering the graph nodes according to their degree, i.e. the number of edges each node has, or partitioning the original graph into subgraphs according to similarity of rows and columns in the respective connectivity matrix. At the end, in one way or the other, these algorithms all test random permutations within the subgraphs in an iterative

way, always assessing the reordering according to a given quality criterion. As a common property, all these algorithms keep the vertex indices as discrete values. This is also true for the graph traversal model, applied by Schlegel et al [43] to order the neurons (N ~ 25 000) of the fly olfactory system along the flow of information. In this approach, neurons were grouped into a sequence of functional layers (<10), corresponding to the mean path length from the sensory periphery to any neuron in the graph while taking into account the connection strengths.

Seung [38] was first to apply graph layout techniques [36,37] to reorder matrices. Defining a quadratic cost function for bandwidth minimization, he derived the gradient, set it to 0 and solved the linear matrix equation by using the Moore-Penrose-inversion. While this approach is fast and works reasonably well to identify connection chains and block-diagonal structures, it is, however, not applicable to other, non-quadratic cost functions, like the one needed to isolate recurrent connections. Here, my approach described above is more general and allows for bandwidth minimization as well as for isolation of recurrent synapses, with only a small excess of recurrent connections compared to the original circuit. The approach also differs from Seung [28] by having two cost functions: a data-dependent and a data-independent one. The first one defines the real objective of the minimization, which can be the total length of all connections of the circuit or the number of recurrent connections. The second term is data-independent because it applies to all circuits, whatever their connectivity, but is specific to the method, i.e. to represent the locations of each node as independent axes in a high-dimensional parameter space. In this representation of the problem, besides going from integer to real values for the positions, the space of solutions is only a subspace of the search space: From all combinations of positions, only those are solutions which represent permutations of the numbers from 1 to N. In other words, each single position has to be different from all other ones. This is achieved by the second cost function, called the 'Pauli-term'.

To gain some intuition, both cost functions, together with their gradients, are shown as a function of positions z_2 and z_3 for the example circuit in Fig 1, with all other positions set to their optimal values, i.e. 0, 1, 4, and 5 (Fig 8A–8F). The recurrency cost is minimal for $z_2 = 2$ and $z_3 = 3$ (Fig 8A). A 1-dimensional path along the diagonal shows a steep decline towards the minimum and a rise to a plateau thereafter (Fig 8B). The Pauli-cost has two minima, one for $z_2 = 2$ and $z_3 = 3$, the other for $z_2 = 3$ and $z_3 = 2$ (Fig 8C). The path along the diagonal shows a steep decline towards the first minimum and steep rise after the second minimum, with a slight energy barrier in between (Fig 8D). In general, the Pauli-cost appears like a bent box of eggs, with small separations between those solutions that represent permutations of positions. The sum of both cost functions again has a single minimum (Fig 8E and 8F), but the plateau in the lower right has a higher gradient towards the minimum than the recurrency term alone.

It is important to note that the Pauli-term only alleviates finding the optimum ordering of the connectivity matrix: in itself, it does not contribute to the quality of the result since taking the arguments of the rank-sorted positions guarantees each position to be an integer value and to be different from any other one anyway. For this reason, also, the optimal balance between the two terms of the cost function cannot be found by treating the relative weight of the Pauli-term as an additional parameter in the gradient descent since it will always lead to zero weight, or the value set by the boundary condition of the parameter, respectively. I, therefore, set the relative weights of the two parts of the cost function empirically. As is shown in Fig 8G and 8H), the relative weight can vary over a broad range.

For an application of the smooth-index algorithm to real connectomes, it might be desirable to include the synaptic weights at each entry of the connectivity matrix. This can be achieved in two different ways. Most easily, the strength of connections can be preserved by sorting the

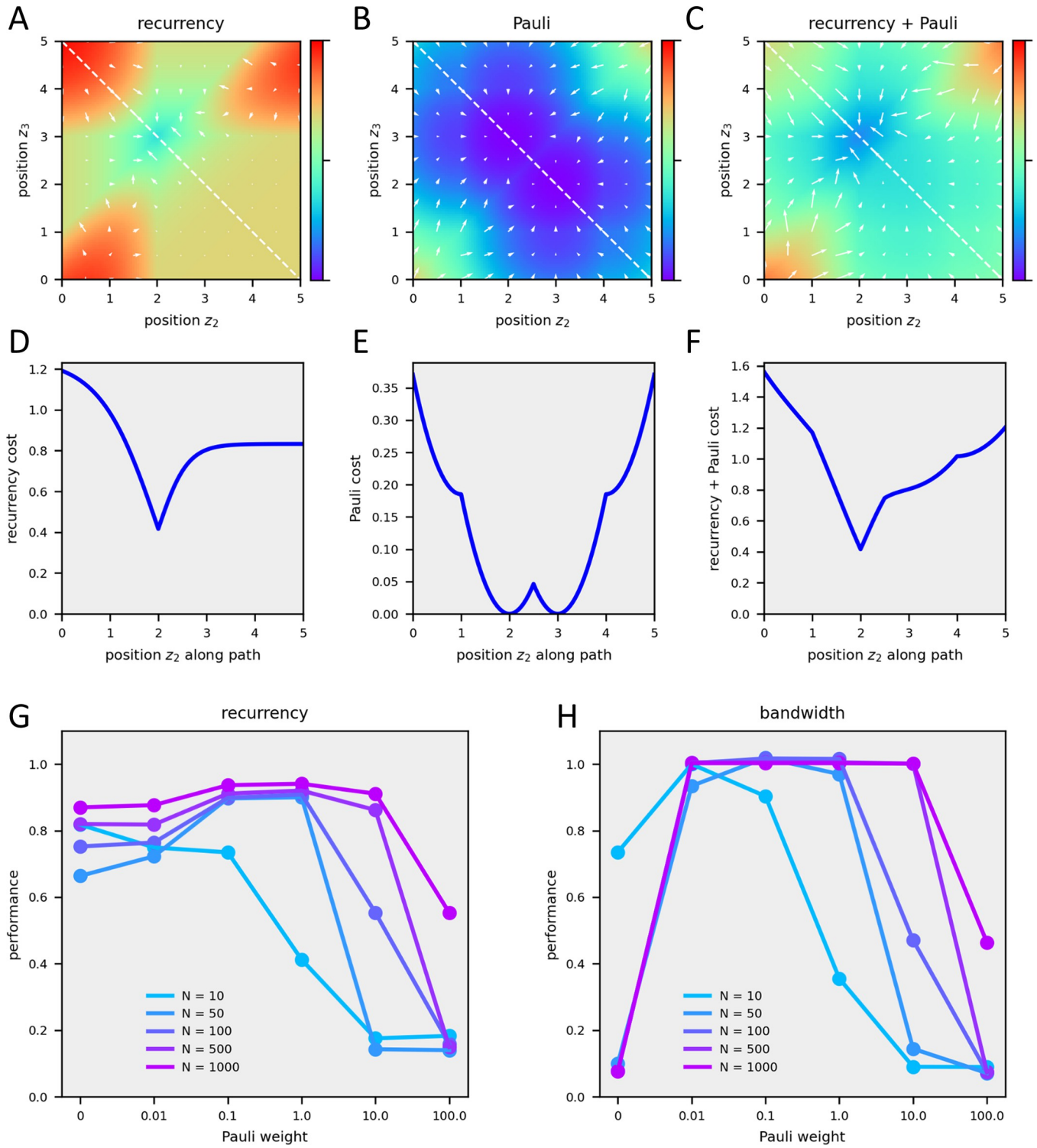

Fig 8. Visualization and balancing the cost function with the Pauli term. **A** Recurrency cost as as f(z_2,z_3). **B** Recurrency cost along the dashed line in A. **C** Pauli term as f(z_2,z_3). **D** Pauli term along the dashed line in C. **E** Recurrency plus Pauli cost as as f(z_2,z_3). **F** Recurrency plus Pauli cost along the dashed line in E. **G** Performance of recurrency minimization as a function of the Pauli term weight. **H** Performance of bandwidth minimization as a function of the Pauli term weight.

https://doi.org/10.1371/journal.pcbi.1011904.g008

binary connectivity matrix and applying the resulting index list to permutate the original connectivity matrix (Fig 6). Alternatively, the synaptic weights of the original connectivity matrix can enter the cost function by giving stronger connections more weight, this way affecting the sorting outcome itself. Future work will show to what extent my approach is generalizable to other problems in the field of matrix reordering as well.

Material and methods

1. Formulas of cost functions and their gradients

In the following, z_m and z_n denote the positions z of a neuron m presynaptic to a neuron n, respectively.

I define $\Delta z_{n,m}$ as: $\Delta z_{n,m} = z_n - z_m + 1$

The Heaviside function H(x) as: $H(x) = \begin{cases} 1 \text{ if } x \geq 0 \\ 0 \text{ if } x < 0 \end{cases}$

The logistic function $\alpha(x, N)$ as: $\alpha(x) = \frac{1}{1+e^{-10x/N}}$

To minimize recurrent connections, I define the length term $L(z)$ as:

$$L(z) = \frac{1}{\sum M} \sum\nolimits_{n \in N} \sum\nolimits_{m \in N} M_{n,m} \left[\alpha(\Delta z_{m,n}) - \frac{1}{2} \right] H(\Delta z_{m,n}) \tag{1}$$

Using $\alpha'(x) = \alpha(x)(1 - \alpha(x))$, the Jacobian of $L(z)$ becomes:

$$\frac{\partial L(z)}{\partial z_n} = \frac{10}{N} \left[-\sum\nolimits_{k \in N} M_{n,k} \alpha'(\Delta z_{k,n}) H(\Delta z_{k,n}) + \sum\nolimits_{k \in N} M_{k,n} \alpha'(\Delta z_{n,k}) H(\Delta z_{n,k}) \right] \tag{2}$$

To minimize the bandwidth, I define the length term $L(z)$ as:

$$L(z) = \frac{1}{N^2} \frac{1}{\sum M} \sum\nolimits_{n \in N} \sum\nolimits_{m \in N} M_{n,m} (\Delta z_{m,n})^2 \tag{3}$$

The Jacobian of $L(z)$ becomes:

$$\frac{\partial L(z)}{\partial z_n} = \frac{1}{N^2} \frac{2}{\sum M} \left[-\sum\nolimits_{k \in N} M_{n,k} \Delta z_{k,n} + \sum\nolimits_{k \in N} M_{k,n} \Delta z_{n,k} \right] \tag{4}$$

In addition, I define the Pauli term $P(z)$ as:

$$P(z) = \frac{1}{N^3} \sum\nolimits_{n \in N} (z_n - rank(z_n))^2 \tag{5}$$

The Jacobian of $P(z)$ becomes:

$$\frac{\partial P(z)}{\partial z_n} = \frac{2}{N^3} (z_n - rank(z_n)) \tag{6}$$

The total cost function $C(z)$ is defined as: $C(z) = L(z) + P(z)$

2. Continuity and differentiability of the Pauli term

In those regions where the positions remain in the same order, i.e. their ranks do not change, the Pauli function is the sum of quadratic functions of the positions, offset by their respective ranks. It is therefore sufficient to analyze continuity and differentiability at the switching points, i.e. where the change of one position z_i leads to a change of the rank of z_i as well as of its neighbor z_j. All other components of the function (denoted as K) remain unaltered. I denote the rank

(z_i) as r, the rank of its larger neighbor z_j then becomes r+1. Note that after switching, i.e. when $z_i > z_j$, the rank of z_i becomes r+1, the one of z_j becomes r. In the following,

$P(z) = \frac{1}{N^3}\sum_{n\in N}(z_n - rank(z_n))^2$ simplifies to:

$$P(z) = (z_i - r)^2 + (z_j - (r+1))^2 + K$$

Continuity:

Left approach:

$$\lim_{\varepsilon\to 0} P(z_i = z_j - \epsilon) = \lim_{\varepsilon\to 0}((z_j - \epsilon - r)^2 + (z_j - (r+1))^2 + K) =$$

$$\lim_{\varepsilon\to 0}((z_j - r - \epsilon)^2 + (z_j - r - 1)^2 + K) = 2(z_j - r)^2 - 2(z_j - r) + 1 + K$$

Right approach:

$$\lim_{\varepsilon\to 0} P(z_i = z_j + \epsilon) = \lim_{\varepsilon\to 0}((z_j + \epsilon - (r+1))^2 + (z_j - r)^2 + K) =$$

$$\lim_{\varepsilon\to 0}((z_j - r + \epsilon - 1)^2 + (z_j - r)^2 + K) = 2(z_j - r)^2 - 2(z_j - r) + 1 + K$$

Since left and right approach leads to the same value, the Pauli function is continuous at the switching points.

Differentiability:

Left approach:

$$\lim_{\varepsilon\to 0}\frac{\partial P(z_i = z_j - \epsilon)}{\partial z_i} = \lim_{\varepsilon\to 0} 2(z_j - \epsilon - r) = 2(z_j - r)$$

Right approach:

$$\lim_{\varepsilon\to 0}\frac{\partial P(z_i = z_j + \epsilon)}{\partial z_i} = \lim_{\varepsilon\to 0} 2(z_j + \epsilon - (r+1)) = 2(z_j - r) - 2$$

Since left and right approach leads to different values, the Pauli function is not differentiable at the switching points.

3. Matrix generation

Original matrices, i.e. before scrambling, for the data shown in Fig 2 were constructed in the following way: elements in the lower triangle were set to 1 with a probability = 0.5, in the upper triangle with a probability = 0.04, elements along the first subdiagonal with a probability = 1. For the data in Fig 3, stripe and block matrices were constructed such that each element within the stripe or block was set to 1 with a probability = 0.5. The stripe width was set to be 40%, the block size was set to be 25% of the matrix size N.

4. Summary description of the smooth-index algorithm

I first turn each vertex index from a discrete integer number into a smooth real number z_i indicating the respective vertex position on a one-dimensional axis. Next, each vertex position is considered to be a parameter value along an independent axis. Thus, the cost becomes a function of an N-dimensional parameter space. This transition brings the advantage that the cost function can become differentiable, allowing gradient descent methods to be applied to search

for a minimum. I applied this idea to reorder matrices such as a) to isolate recurrent connections, and b) to minimize the bandwidth.

a) Recurrency minimization: A recurrent connection is characterized by the fact that the position z_m of a presynaptic neuron m is larger than the position z_n of a postsynaptic neuron n. However, the number of recurrent connections varies in a discrete step at the point where $z_m = z_n$, and, hence, is not a differentiable function of z. I therefore chose the average length $z_m - z_n$ of all recurrent connections as a proxy for the number of recurrent connections. Taking a saturating function of the length reduced the stronger influence of longer connections over shorter ones. The cost function and its gradient are formulated by Eq 1 and Eq 2, respectively.

b) Bandwidth minimization: Here, the cost is defined as the mean of the squared length of all connections, irrespective of their sign. The respective cost function and its gradient are formulated by Eq 3 and Eq 4, respectively.

c) The Pauli term: To keep all positions from collapsing into a single value, I define an additional Pauli term as the mean of the squared differences between the positions and their rank within the sorted position array. This part of the cost function and its gradient are formulated by Eq 5 and Eq 6, respectively.

The total cost is defined as the sum of the specific cost function and the Pauli term.

Both the total cost function as well as its gradient (Eqs 1,2,5,6 for minimizing recurrent connections, Eqs 3,4,5,6 for minimizing bandwidth) are passed onto the Python Scipy minimize function [41] using the bound constrained method of Broyden, Fletcher, Goldfarb, and Shanno ('L-BFGS-B'). To return to discrete vertex indices, as the final step, the permutation list π that reorders the matrix is obtained as the vector containing the arguments of the rank-sorted parameters at the minimum of the cost function. Termination of the search is set by the tolerance value used by the Scipy minimize function. This value is set in the 'SI_sort' function of the library to different values depending on the size of the circuit. The default value is 10^{-8} and decreases for larger circuits down to 10^{-11}.

Acknowledgments

I am grateful to Winfried Denk for pointing out the 'integer relaxation' technique to me. I thank him, Christian Leibold, Sebastian Seung and Juergen Haag for critically reading earlier versions of the manuscript and for many discussions and suggestions. Thanks also to Srinivasan Turaga and Janne Lappalainen for providing me with the intra-columnar connectivity matrix used in Fig 6.

Author Contributions

Conceptualization: Alexander Borst.

Formal analysis: Alexander Borst.

Methodology: Alexander Borst.

Project administration: Alexander Borst.

Software: Alexander Borst.

Validation: Alexander Borst.

Visualization: Alexander Borst.

Writing – original draft: Alexander Borst.

Writing – review & editing: Alexander Borst.

References

1. Abbott LF, Bock DD, Callaway EM, Denk W, Dulac C, Fairhall AL, et al. (2020) The mind of a mouse. Cell 182:1372–1376. https://doi.org/10.1016/j.cell.2020.08.010 PMID: 32946777
2. Bargmann C, Marder E (2013) From the connectome to brain function. Nature Methods 10: 483–490. https://doi.org/10.1038/nmeth.2451 PMID: 23866325
3. Briggman KL, Helmstaedter M, Denk W (2011) Wiring specificity in the direction-selectivity circuit of the retina. Nature 471:183–188. https://doi.org/10.1038/nature09818 PMID: 21390125
4. Denk W, Horstmann H (2004) Serial block-face scanning electron microscopy to reconstruct three dimensional tissue nanostructure. PLoS Biol 2:e329. https://doi.org/10.1371/journal.pbio.0020329 PMID: 15514700
5. Helmstaedter M, Briggman KL, Turaga SC, Jain V, Seung HS, Denk W (2013) Connectomic reconstruction of the inner plexiform layer in the mouse retina. Nature 500:168–174. https://doi.org/10.1038/nature12346 PMID: 23925239
6. Lichtman JW, Denk W (2011) The big and the small: challenges of imaging the brain's circuits. Science 334:618–623. https://doi.org/10.1126/science.1209168 PMID: 22053041
7. Markram H, Muller E, Ramaswamy S, Reimann MW, Abdellah M, Sanchez CA, Ailamaki A, Alonso-Nanclares L, Antille N, Arsever S, et al. (2015) Reconstruction and simulation of neocortical microcircuitry. Cell 163:456–492. https://doi.org/10.1016/j.cell.2015.09.029 PMID: 26451489
8. Motta A, Berning M, Boergens KM, Staffler B, Beining M, Loomba S, et al. (2019) Dense connectomic reconstruction in layer 4 of the somatosensory cortex. Science 366:eaay3134. https://doi.org/10.1126/science.aay3134 PMID: 31649140
9. Shinomiya K, Huang G, Lu Z, Parag T, Xu CS, Aniceto R, et al. (2019) Comparisons between the ON- and OFF-edge motion pathways in the Drosophila brain. Elife 8:e40025. https://doi.org/10.7554/eLife.40025 PMID: 30624205
10. Takemura SY, Nern A, Chklovskii DB, Scheffer LK, Rubin GM, Meinertzhagen IA (2017) The comprehensive connectome of a neural substrate for 'ON' motion detection in Drosophila. Elife 6:e24394. https://doi.org/10.7554/eLife.24394 PMID: 28432786
11. Dorkenwald S et. al. (2022) FlyWire: online community for whole-brain connectomics. Nature Methods 19: 119–128. https://doi.org/10.1038/s41592-021-01330-0 PMID: 34949809
12. Abeles M, Bergman H, Margalit E, Vaadia E (1993) Spatiotemporal firing patterns in the frontal cortex of behaving monkeys. J Neurophysiol 70: 1629–1638. https://doi.org/10.1152/jn.1993.70.4.1629 PMID: 8283219
13. Diesmann M, Gewaltig M-O, Aertsen A (1999) Stable propagation of synchronous firing in cortical neural networks. Nature 402: 529–533.
14. Strang G (2009) Introduction to linear algebra. Wellesley-Cambridge Press, Wellesley, MA, fourth edition.
15. Dayan P, Abbott LF (2001) Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems. The MIT Press. 16
16. Christodoulou G, Vogels T (2022) The eigenvalue value (in neuroscience). OSF Preprints 10.31219/osf.io/evqhy.
17. Borst A, Leibold C (2023) Connecting connectomes to physiology. J Neurosci 43: 3599–3610. https://doi.org/10.1523/JNEUROSCI.2208-22.2023 PMID: 37197984
18. Seung HS (1996) How the brain keeps the eyes still. Proc Natl Acad Sci USA 93: 13339–13344. https://doi.org/10.1073/pnas.93.23.13339 PMID: 8917592
19. Seung HS, Lee DD, Reis BY, Tank DW (2000a) Stability of the memory of eye position in a recurrent network of conductance-based model neurons. Neuron 26:259–271. https://doi.org/10.1016/s0896-6273(00)81155-1 PMID: 10798409
20. Usher M, McClelland JL (2001) The time course of perceptual choice: the leaky, competing accumulator model. Psychol Rev 108:550–592. https://doi.org/10.1037/0033-295x.108.3.550 PMID: 11488378
21. Wehner R (2003) Desert ant navigation: how miniature brains solve complex tasks. J Comp Physiol A 189: 579–588. https://doi.org/10.1007/s00359-003-0431-1 PMID: 12879352
22. Wittlinger M, Wehner R, Wolf H (2006) The ant odometer: stepping on stilts and stumps. Science 312: 1965–1967. https://doi.org/10.1126/science.1126912 PMID: 16809544
23. Borst A, Haag J, Mauss AS (2020) How fly neurons compute the direction of visual motion. J Comp Physiol A 206:109–124. https://doi.org/10.1007/s00359-019-01375-9 PMID: 31691093
24. Borst A, Groschner LN (2023) How flies see motion. Ann Rev Neurosci 46:17–37. https://doi.org/10.1146/annurev-neuro-080422-111929 PMID: 37428604

25. Cuthill E, McKee J (1969) Reducing the bandwidth of sparse symmetric matrices. In: Proceedings of the 1969 24th National Conference, ACM '69, p. 157–172, New York, NY, USA. Association for Computing Machinery.
26. Tarjan RE (1972) Depth-first search and linear graph algorithms. SIAM J. Comput. 1:146–160.
27. Tarjan RE (1974) Testing flow graph reducibility. J. Comput. Syst. Sci. 9:355–365.
28. Eades P, Lin X, Smyth WF (1993) A fast and effective heuristic for the feedback arc set problem. Information Processing Letters 47: 319–323.
29. Hassin R, Rubinstein S (1994) Approximations for the maximum acyclic subgraph problem. Information Processing Letters 51: 133–140.
30. Brandenburg FJ, Hanauer K (2011) Sorting heuristics for the feedback arc set problem. University Passau, Dept of Informatics & Mathematics, Technical Report, MIP-1104.
31. Baharev A, Schichl H, Neumaier A, Achterberg T (2021) An exact method for the minimum feedback arc set problem. ACM J of experimental algorithms 26 (1): Article 1.4.
32. Geladaris V, Lionakis P, Tollis IG (2022) Computing a feedback arc set using pagerank. arXiv:2208.09234v2.
33. Page L, Brin S, Motwani R, Winograd T (1999) The page-rank citation ranking: Bringing order to the web. Technical Report 1999–66, Stanford InfoLab Previous number = SIDL-WP-1999-0120.
34. Liiv I (2010) Seriation and matrix reordering methods: An historical overview. Stat. Anal. Data Min. 3: 70–91.
35. Behrisch M, Bach B, Henry Riche N, Schreck T, Fekete JD (2016) Matrix reordering methods for table and network visualization. Computer Graphics Forum 35:693–716.
36. Sugiyama K, Misue K (1995) A simple and unified method for drawing graphs: Magnetic-spring algorithm. Proc Graph Drawing (GD '94), 364–375.
37. Carmel L, Harel D, Koren Y (2004). Combining hierarchy and energy for drawing directed graphs. IEEE Trans. Vis Comput Graph 10: 46–57. https://doi.org/10.1109/TVCG.2004.1260757 PMID: 15382697
38. Seung HS (2009) Reading the book of memory: sparse sampling versus dense mapping of connectomes. Neuron 62: 17–29. https://doi.org/10.1016/j.neuron.2009.03.020 PMID: 19376064
39. Vogel F, Jenatton R, Bach F, d'Aspremont A (2013) Convex relaxations for permutation problems. Advances in Neural Information Processing Systems (NIPS) 26.
40. Vogelstein JTet. al. (215) Fast approximate quadratic programming for graph matching. PLoS ONE, https://doi.org/10.1371/journal.pone.0121002 PMID: 25886624
41. Virtanen P, Gommers R, Oliphant TE, Haberland M, Reddy T, Cournapeau D, Burovski E, et al. (2020) SciPy 1.0: Fundamental algorithms for scientific computing in python. Nature Methods 17: 261–272. https://doi.org/10.1038/s41592-019-0686-2 PMID: 32015543
42. Ausiello G, D'Atri A, Protasi M (1980) Structure preserving reductions among convex optimization problems. J Computer System Sciences, 21:136–153.
43. Schlegel P, Bates AS, Stuerner T, Jagannathan SR, Drummond N, Hsu J, et al. (2021) Information flow, cell types and stereotypy in a full olfactory connectome. eLife 2021; 10:e66018. https://doi.org/10.7554/eLife.66018 PMID: 34032214