Supplementary information for "Modular representation and control of floppy networks"

Siheng Chen, Fabio Giardina, Gary P. T. Choi, L. Mahadevan

S1 Sparse null space decomposition

S1.1 Overview of the algorithm

In this section, we provide the review and development of algorithms for solving the sparse null space decomposition (SND) [1]. We first introduce the ABS class of algorithms proposed by Abaffy, Broyden and Spedicato [2, 3]. The ABS class of algorithms are iterative algorithms for solving the linear system $Ax = b$ where $A \in \mathbb{R}^{m\times n}$ has full row rank and $b \in \mathbb{R}^m$. Let $A = [a_1, a_2, \ldots, a_n]$, where each a_i is an $m \times 1$ column vector. We start with an arbitrary initial vector $x_1 \in \mathbb{R}^n$ and an arbitrary nonsingular matrix $H_1 \in \mathbb{R}^{n\times n}$, e.g. the identity matrix. At the i-th step, suppose we already have a solution of the first $i-1$ equations of $Ax = b$ (denoted as x_i) and an an $n \times n$ matrix with rows generating the null space of the first $i-1$ rows of A (denoted as H_i, also called the Abaffian matrix). We get x_{i+1} and H_{i+1} from x_i and H_i using the formulas below:

$$x_{i+1} = x_i + \alpha_i q_i, \tag{S1}$$

where α_i is the step size, $q_i = H_i^T z_i$, z_i is some search vector, and

$$H_{i+1} = H_i - \frac{H_i a_i w_i^T H_i}{w_i^T H_i a_i}, \tag{S2}$$

where w_i (called the Abaffy's parameter) satisfies $w_i^T H_i a_i \neq 0$. By iteratively updating x_i and H_i, we finally obtain x_{m+1} (a solution of the entire system $Ax = b$) and H_{m+1} (a matrix with rows generating the null space of the entire matrix A).

Chen et al. [4] proposed a generalization of the ABS algorithms, called the Extended ABS (EABS) algorithms. The difference between the ABS algorithms and the EABS algorithms is in the procedure of updating the Abaffian matrices H_i. Unlike the ABS algorithms, the EABS algorithms update H_i by

$$H_{i+1} = G_i H_i, \tag{S3}$$

where G_i is a matrix with $G_i x = 0$ if and only if $x = \lambda H_i a_i$ for some λ.

The SND algorithm in [1] utilizes the EABS algorithm to find a sparse null space basis. The idea is to carefully set the matrices G_i such that they satisfy the requirement in EABS and make the resulting H_{m+1} as sparse as possible. At the i-th iteration, let $A_{m-i} = (a_i, \ldots, a_m)^T$ and $\widetilde{A}_i = A_{m-i} H_i^T$. Also, let r_t and c_j be the number of nonzero elements in row t and column j in $\widetilde{A}_i$. The SND algorithm uses the Markowitz pivot selection criterion [5] and finds the (t, j) entry of $\widetilde{A}_i$ such that $(r_t - 1)(c_j - 1)$ is minimal. Using the entry as a pivot element, the algorithm sets G_i to perform an elimination similar to the Gaussian elimination and obtain H_{i+1}. In other words, the algorithm selects a nonzero entry in the current submatrix, and brings it to the position (i, i) by a matrix operation in such a way that there are as few nonzero entries in the i-th row and i-th column in the resulting H_{i+1} as possible. By

iteratively updating x_i and H_i, the final matrix H_{m+1} gives a sparse null space basis. More details of the computational procedure can be found in [1].

Note that the Markowitz pivoting method used in the SND algorithm is local in the sense that it tries to minimize the number of nonzero entries at each step separately, without considering the possibility that some combinations of suboptimal pivots in several steps may further reduce the number of nonzero entries.

Our mode decomposition problem gives a physical interpretation of the SND algorithm. More specifically, the algorithm looks for the most separated element when selecting the pivot at each iteration and finally achieves a sparse null space basis. In a physical network, this corresponds to a decomposition of the floppy modes that maximizes hierarchy and separability.

S1.2 Implementation

The rigidity matrix is constructed from the geometric constraints as described in the main text, and the fixed nodes constraints described in the Methods. The rows are shuffled every time before applying the SND method. The resulting mode decomposition might be different each time, so we present the average results (for example, in main text Fig. 2**c** and 2**f**).

S1.3 Comparison with other methods

We provide a more detailed comparison between the SND method (referred to as Method I) and other methods for generating null space bases. As discussed in the main text, given a matrix A, the MATLAB's built-in function `null` computes an orthonormal basis for the null space of A using the singular value decomposition (SVD) algorithm (referred to as Method II).

An alternative numerical approach for finding a null space basis for A is to apply the MATLAB's `null` function on the matrix A^TA (referred to as Method III). Note that for any $x \in \text{null}(A)$, we have $A^TAx = A^T(0) = 0$ and hence $x \in \text{null}(A^TA)$. Therefore we have $\text{null}(A) \subset \text{null}(A^TA)$. Also, for any $y \in \text{null}(A^TA)$, we have $A^TAy = 0 \Rightarrow y^TA^TAy = 0 \Rightarrow (Ay)^T(Ay) = 0 \Rightarrow Ay = 0$ and hence $y \in \text{null}(A)$, which implies that $\text{null}(A^TA) \subset \text{null}(A)$. This shows that $\text{null}(A) = \text{null}(A^TA)$. While the two null spaces are identical, the bases obtained numerically using $\texttt{null}(A)$ and $\texttt{null}(A^TA)$ are not necessarily the same.

Another method for finding a null space basis for A is to use the QR factorization (referred to as Method IV). By applying the QR factorization on A^T, we obtain $A^T = QR$ where Q is a $m \times m$ orthogonal matrix and R is a $m \times n$ upper triangular matrix. More explicitly, if $\dim(\text{null}(A)) = m - r$, we have

$$A^T = QR = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix} \begin{pmatrix} R_1 \\ 0 \end{pmatrix}, \tag{S4}$$

where $Q_1 \in \mathbb{R}^{m\times r}$, $Q_2 \in \mathbb{R}^{m\times(m-r)}$, $R_1 \in \mathbb{R}^{r\times n}$, and $0 \in \mathbb{R}^{(m-r)\times n}$. Now, we have

$$\begin{pmatrix} R_1 \\ 0 \end{pmatrix} = R = Q^TQR = Q^TA^T = \begin{pmatrix} Q_1^TA^T \\ Q_2^TA^T \end{pmatrix}, \tag{S5}$$

and hence $0 = Q_2^TA^T = AQ_2$. Since Q_2 contains exactly $m - r$ columns, it follows that the columns of Q_2 form a null space basis for A.

Fig. 2 in the main text shows the mode decomposition of an example triangular network using the SND method and the SVD method. Here we adopt a larger network, and compare the mode decomposition between the above four methods (Fig. S1**a**–**d**). Again, the SND produces a decomposition which is easy to interpret, as the motions are all separated. As noted in the main text, all the representations are equivalent. Modes in one set of representation can be decomposed into a linear combination of modes in another set. Here, the modes generated by Method II, III, and IV are decomposed into the modes

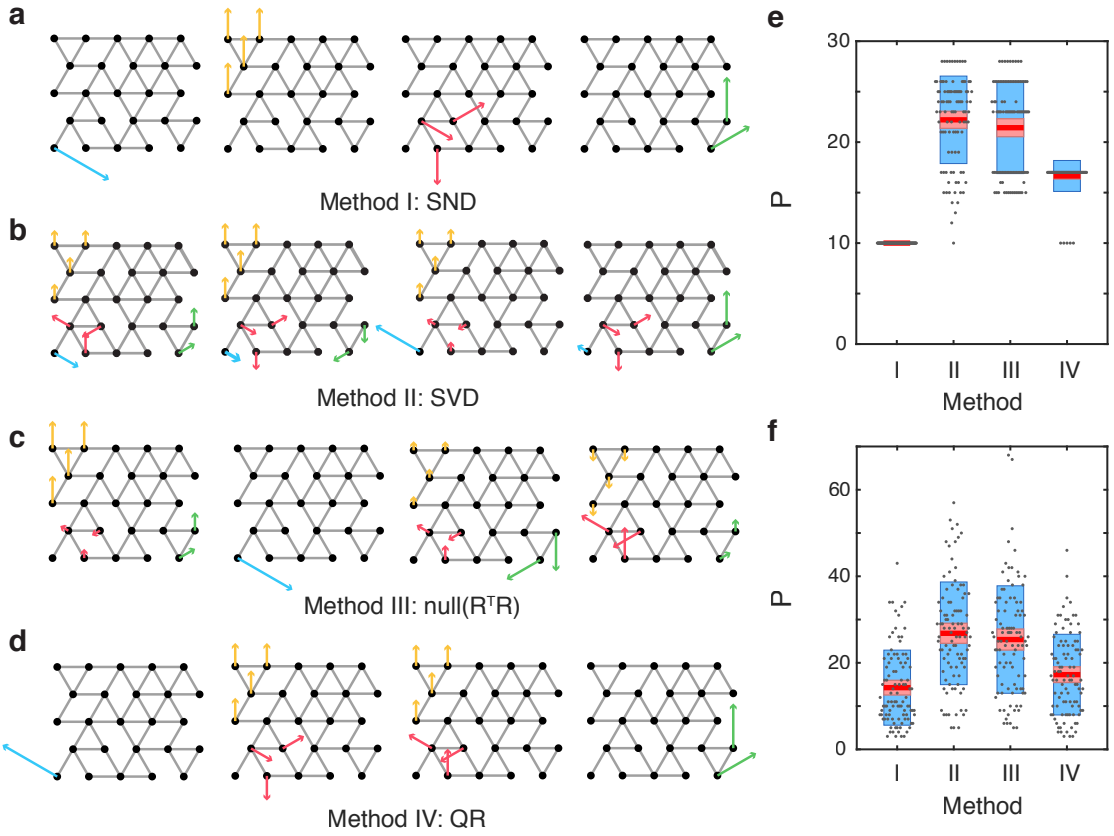

Figure S1: **Mode decomposition of a 5×5 network using four different methods.** **a**, The SND method on rigidity matrix $\boldsymbol{R}$. **b**, The SVD method on $\boldsymbol{R}$. **c**, The SVD method on $\boldsymbol{R}^T\boldsymbol{R}$. **d**, The QR decomposition on $\boldsymbol{R}$. The modes in methods II-IV are decomposed to linear combination of those in I and colored accordingly. **e**, The participation rate P of modes for the four methods and 100 repeats, on this 5×5 network. **f**, The participation rate of modes for the four methods and 100 repeats on networks with link density $\rho = 43/56$.

in Method I, and are colored accordingly. All the modes in Method II, III, and IV are not spatially separating the motion in different areas, thus harder to interpret physically. The participation rate P is also much lower in method I compared to that in Method II, III, and IV. Fig. S1**e** shows the distribution of P for modes calculated from the four methods with the rows of rigidity matrix randomly shuffled 100 times. Fig. S1**f** shows the distribution of P for modes calculated from the four methods on 100 different networks with the same link density.

One may also be interested in comparing the computational efficiency of the four above-mentioned methods. Here we apply the four methods on randomly generated networks with different number of nodes (denoted by n) for null space basis generation. For each n, we repeat the experiment for 100 times and record the average time taken (denoted by t). Fig. S2 shows the log-log plot of the time taken for networks with different size, from which it can be observed that SND is the fastest among all four methods. This shows that SND is not only more advantageous in terms of producing a sparse representation but also in computational efficiency.

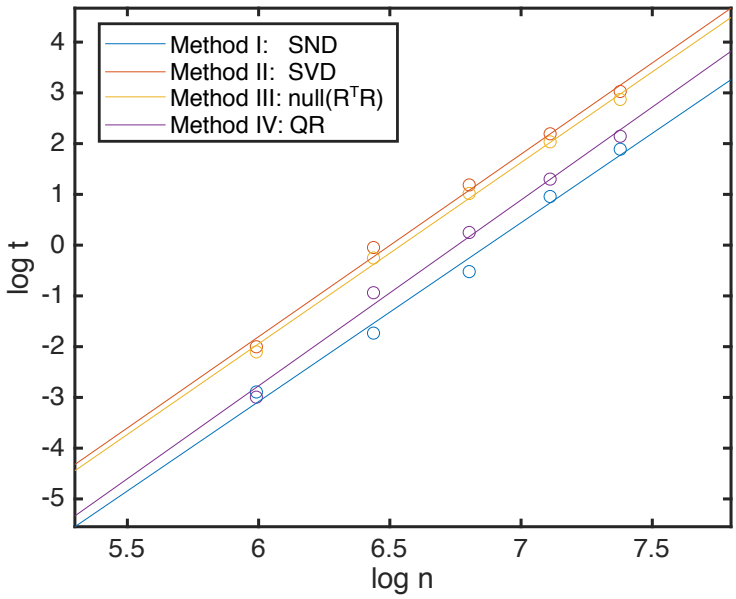

Figure S2: **Computational time of null space basis generation using different methods.** Here n is the number of nodes and t is the average time taken (in second) for the 100 randomly generated networks. For each method, a least-squares line is superimposed using the MATLAB `lsline` function.

S1.4 Other applications of SND algorithm

S1.4.1 Identify modes in larger networks

The modes in Fig. S1 might be easy to identify through observation. However, in larger networks, it is more difficult to identify all the modes just by inspecting the structures by eye. In a larger network constructed in Fig. S3 (inspired by the network in [6]), there are four floppy modes, only one of which is easy to identify through observation (the rotation of a single node shown in the leftmost figure in Fig. S3a. By applying the SND protocol to the rigidity matrix, we can identify all four modes, shown in a hierarchical (left two modes) and spatially separated (right two modes) manner. On the other hand, the SVD method mixes all these easily-interpretable modes and produce another set of motions (Fig. S3**b**).

S1.4.2 Mode Classification

It has been proved that, if the rigidity matrix of a network has full rank, the mode is a finite mode [7], which means that the infinitesimal motion can be extended to a finite amount. An example would be a horizontal rod with the left end fixed. With 3 constraints and 4 coordinates (the left node x_1, y_1 and the right node x_2, y_2), with both x_1 and y_1 spatially fixed, we have

$$\boldsymbol{R} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \end{pmatrix}, \tag{S6}$$

which is full rank. The dimension of the null space is 1, and a null space basis vector is $(0, 0, 0, 1)$, which corresponds to a finite rotation.

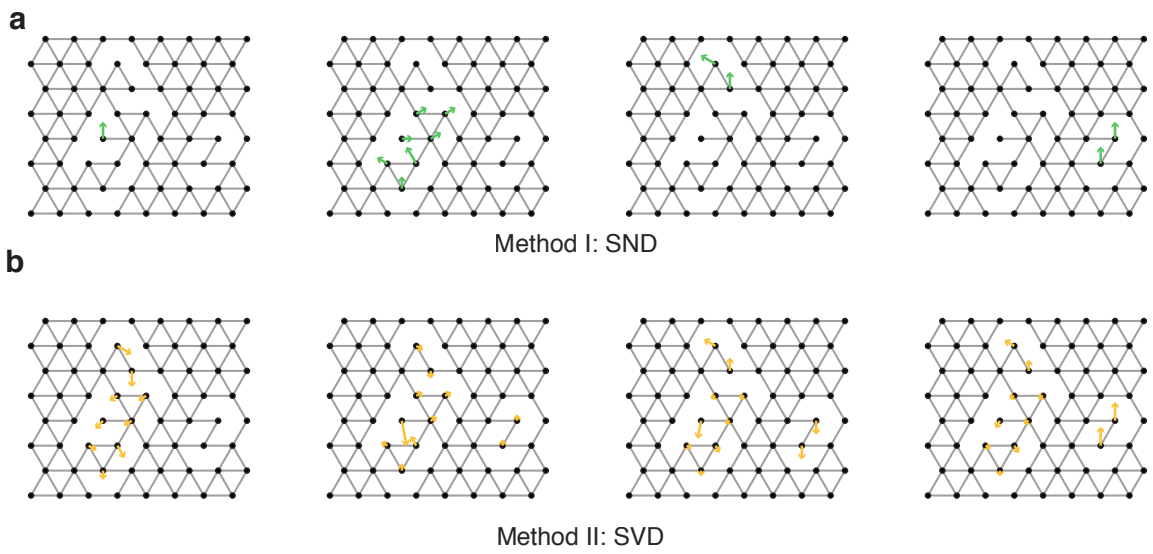

Figure S3: **Identifying modes in a larger network using the SND method and the SVD method. a**, In this larger network, it is more difficult to identify the floppy modes through observation. Applying the SND method, the four modes are identified in a hierarchical and spatially separated way. **b**, The SVD method can also identify four modes, but it is hard to separate out the actuation at different locations.

However, the opposite is not true. If the rigidity matrix does not have full rank, there are three possible scenarios listed as follows, and as shown in Fig. S4**a**.

Infinitesimal mode The mode could be an infinitesimal (but not finite) mode. An example of this is in Fig. S4**b**. Assume that the left and the right node are both fixed, and the coordinates are $(x_1, y_1, x_2, y_2, x_3, y_3)$ from left to right. The rigidity matrix is

$$\boldsymbol{R} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 \end{pmatrix}. \tag{S7}$$

The rank of this matrix is 5 (less than 6), and the mode is indeed only infinitesimal.

Finite mode An example of the rigidity matrix being not full-rank, but still having a finite mode, is an over-constrained network with only rotational modes (e.g. the structure in Fig. S4**c**). There is a finite rotational mode while the rigidity matrix does not have full rank, because there is a redundant constraint. There is one diagonal link being redundant.

Both finite and infinitesimal modes It is also possible to create a mixture of the two modes. Consider the structure in Fig. S4**d**. The rigidity matrix is still rank-deficient, and the structure has the two modes shown in Fig. S4**b-c**, being infinitesimal and finite respectively.

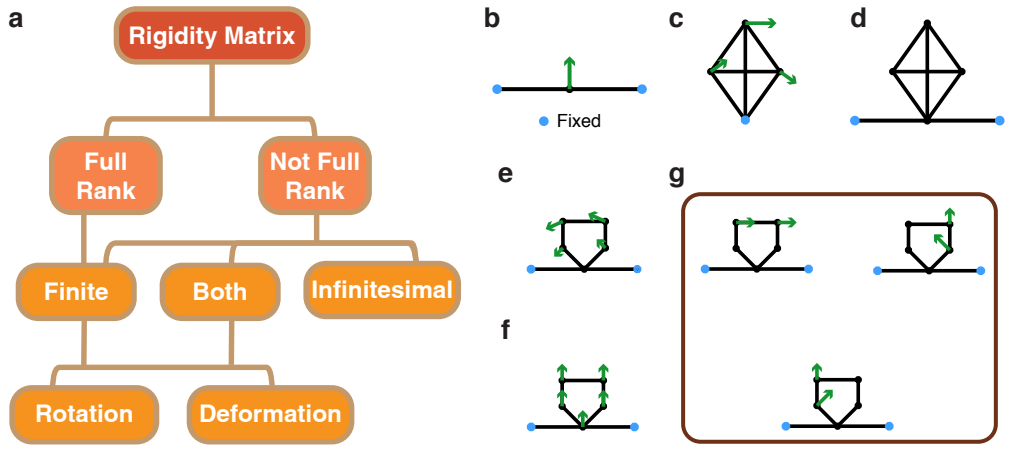

Figure S4: **Mode classification.** **a**, The classification diagram for floppy modes in a mechanical network based on the rank of rigidity matrices. **b**, An example network with rank-deficient rigidity matrix and an infinitesimal mode. **c**, An example network with rank-deficient rigidity matrix and a finite mode. **d**, An example network with rank-deficient rigidity matrix and both finite and infinitesimal modes. **e-g**, An example network with 5 modes, identified by hierarchical SND algorithm (see Section S2). **e** shows the pure rotational mode of the pentagon (finite, rotation). **f** shows the infinitesimal mode. **g** shows the remaining three modes (finite, deformation).

In addition, the finite modes can be classified as rotation and deformation, depending on whether there is relative motion between the nodes. The example in Fig. S4**c** shows the rotation. If we remove the two diagonal links, there will be an additional deformation mode.

An example To illustrate the idea above, we use a simple example as shown in Fig. S4**e**. The two blue nodes are fixed spatially, and all the bonds can rotate freely around joints. If we apply the multi-scale SND algorithm (see next section), we get the hierarchical representation in Fig. S4**e-g**.

If we isolate the pentagon and fix the bottom node of the pentagon, the five constraints are independent and the rigidity matrix has full rank. Therefore, the modes within the pentagon are finite modes (Fig. S4**e** and **g**). The multi-scale SND algorithm yields all the purely rotational mode (Fig. S4**e**), so all the other modes are deformation modes (Fig. S4**g**). Finally, the remaining mode involves the motion of the bottom center node so we have to consider the bottom two links. The rigidity matrix does not have full rank. Comparing it with the mode in Fig. S4**b**, we know that this mode is only an infinitesimal mode.

S2 Multi-scale SND algorithm

S2.1 Identifying bi-connected components

In graph theory, a bi-connected graph is defined as a connected graph where, if any one of the vertices is removed, the graph remains connected. A bi-connected component of a graph is a maximally bi-connected subgraph [8]. For example, in Fig. S5**a**, the graph can be decomposed into 6 bi-connected components, as shown in Fig. S5**b**. Within one bi-connected component, the network remains connected after removal of any vertex. Two bi-connected components are attached to each other at one shared vertex, which is called cut vertex or articulation point. Removal of any of these cut vertices will result in the increase of connected components in the network.

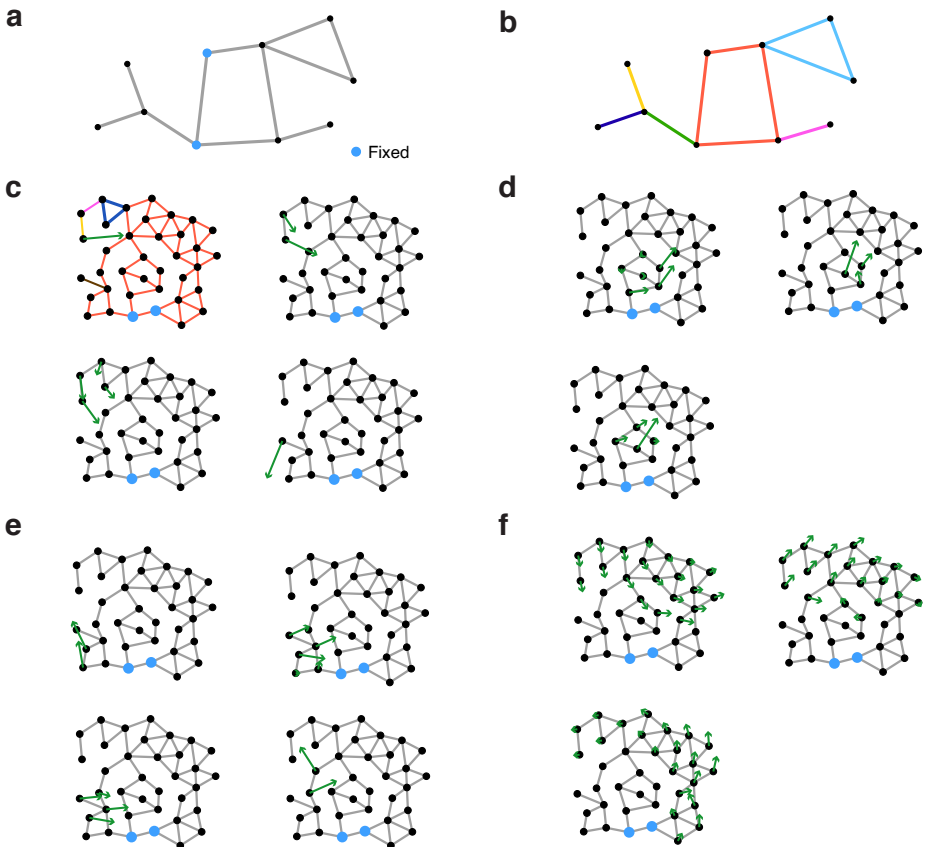

Figure S5: **Applying multi-scale SND to identify all modes in a large network with multiple DOF. a**, A simple network with two fixed nodes. **b**, The bi-connected components in the network. Each color represents one bi-connected component. Two bi-connected components share at most one node. **c-f**, The 14 modes identified by the multi-scale SND algorithm in a network, organized into four groups for easier interpretation. **c**, All pure rotation modes. The bi-connected components are shown in the top left mode. **d**, All modes concerning the chunk at the center of the network. **e**, Hierarchical modes at the bottom left of the network. **f**, All global (larger) modes.

This definition exactly suits our need to identify sections (subgraphs) which can rotate freely around the remaining network, since there is at most one joint vertex between any two bi-connected components. We can apply this concept to identify all the rotational mode before using SND on the subgraphs.

S2.2 Recursive SND algorithm

The procedure for identifying mode representation using multi-scale SND:

(a) Fix two nodes in space.

(b) Identify all the bi-connected components in the graph.

(c) Identify the "base" and the "branches". The bi-connected component containing the fixed nodes are the "base" structure, and each set of connected bi-connected components is considered one branch.

(For example, in Fig. S5**b**, the left three bi-connected components (green, yellow, and purple) only form one branch because they are connected and attached to the base structure (red).)

(d) Add the rotational mode of each branch around the base structure to the collection of all modes.

(e) Apply SND to the "base structure" to find out a set of sparse modes. If any mode in the base structure involves movement of a cut vertex (connecting vertex to a branch), set the motion of all the branch nodes to be exactly the same as the cut vertex.

(f) Apply (b)-(d) recursively to all the branches by treating the cut vertex as a fixed node, and identify all modes in the branch structure.

(g) Add the modes in branches to the full set of modes.

In step (a), since any 2D network has two translation modes and one rotation mode (three rigid body modes), we can always fix two nodes to freeze these modes and identify the remaining modes associated with the relative motion. In the end, we can always add the three rigid body modes back to get the full representation. Adding fixed nodes has the advantage of separating the rigid body modes and the modes with respect to relative motion within the network. Note that there is no additional randomness introduced by this multi-scale approach besides the random shuffling of the rigidity matrix when applying the SND. The modes within a sub-branch might be different from the random shuffling, but it will not involve any nodes outside that branch.

S2.3 Examples

We demonstrate how the procedure above can be applied on two example networks. The first example is in Fig. 2**d** in the main text. The two nodes on the bottom left are fixed. The identification of bi-connected components separates the left and the right sub-graph. Since the left sub-graph contains the fixed nodes, the left sub-graph is the base structure and the yellow sub-graph is a branch. Step (c) will thus add the rotational mode of the branch with respect to the base (mode (1)). Then the algorithm will apply SND to the base structure, and identify the mode (2). Once the modes in the base structure are identified, the algorithm repeats the process above to the branch structure. Since there is only one bi-connected component in the branch, there is no more purely rotational mode. The SND is then applied to identify the modes (3)-(5), which shows a clear hierarchy. Overall the modes identified by SND agrees with our intuition (rotation of the right section), and can even outperform human eyes by identifying the floppiness in a hierarchical manner. Applying SVD directly, in contrast, will result in five modes, each of which involves the motion of almost all nodes that can possibly move.

The second example is a network with many more modes (Fig. S5, 14 modes). The modes are organized in a way such that it is easier to interpret. The identification of bi-connected components is shown in the top left mode in Fig. S5**c**, where the red sub-graph is the base structure, and the remaining consists of two branches. Each branch corresponds to one rotational mode (bottom left and bottom right in Fig. S5**c**). When this process is applied recursively, the top left branch has a sub-branch consisting of the pink and yellow edges - corresponding to the rotational mode as shown in Fig. S5**c** top right. The next level of recursion identifies the last rotational mode (Fig. S5**c**, top left).

The remaining modes are identified from applying SND to the base structure. Fig. S5**d** shows the modes that only concerns the motion of the center chunk consisting of 6 nodes. Fig. S5**e** shows a hierarchical mode decomposition of the modes on the bottom left of the network. Finally, Fig. S5**f** shows the bigger and more global modes. Note that if a mode involves the movement of a cut vertex, all the branch nodes will follow the same movement as the cut vertex. (In Fig. S5**f** all the top left four nodes have exactly the same motion as that of the cut vertex.)

S3 Motion Primitives

S3.1 Extracting motion primitives

Given a random floppy network in a particular initial state, we define the motion primitives as the generalized rotations that generate the zero modes obtained using either SND or SVD in that initial state. Given the node velocities of modes $\boldsymbol{v}_i \in \mathbb{R}^n$, $i \in 1, ..., n-r$, we seek to generalize them to arbitrary configurations of the network by transforming the modes into generalized rotations. n is the number of nodes times the dimension of the problem, r is the rank of the rigidity matrix, N the number of nodes in the network, m the number of constraints, and $\boldsymbol{x} \in \mathbb{R}^n$ is a vector containing all positions of the nodes in the plane. Assume $\mathbf{Q}(\boldsymbol{x}) \in \mathbb{R}^{n \times m}$ is a matrix whose column vectors contain all possible velocities resulting from rotations of connected nodes in the network, and $\boldsymbol{w} \in \mathbb{R}^m$ is a vector containing weights of activation for the rotational mode. The weights that generate mode v_i in the initial configuration are found by

$$\boldsymbol{w}_i = \mathbf{Q}^\dagger(\boldsymbol{x}_0)\boldsymbol{v}_i, \tag{S8}$$

where † denotes the pseudoinverse of the matrix $\mathbf{Q}$. For every state of the network, $\mathbf{Q}$ needs to be updated with the positions of the nodes, however, when multiplied by the weights w_i which are only derived once at t_0, the resulting rotations will correspond to the rotations that generated the zero modes in the initial configuration, i.e. the motion primitives. Note that due to the heuristic nature of SND, the motion primitives are not unique but depend on the pivot selection and order of rows in the supplied rigidity matrix.

S3.2 Equations of motion

As above, the positions of each node is given in $\boldsymbol{x} \in \mathbb{R}^n$. The equations of motion are then given by

$$\mathbf{M}\ddot{\boldsymbol{x}} = \mathbf{R}^T\boldsymbol{\lambda}, \tag{S9}$$

with $\mathbf{M}$ the mass matrix (identity matrix times mass of node if all nodes are equal), $\mathbf{R}$ the rigidity matrix, and $\boldsymbol{\lambda}$ the constraint forces which are given by

$$\boldsymbol{\lambda} = -\left(\mathbf{R}\mathbf{M}^{-1}\mathbf{R}^T\right)^{-1}\dot{\mathbf{R}}\dot{\boldsymbol{x}}, \tag{S10}$$

with $\dot{\mathbf{R}}$ the time derivative of the rigidity matrix (the rigidity matrix is a function of the node positions which change over time in the reaching task).

Along with the dynamical state, we integrate a cost function defined as the sum of the square of all the nodes' velocities integrated over time

$$C = \int_0^{t_F} \dot{\boldsymbol{x}}^T\dot{\boldsymbol{x}}\, dt. \tag{S11}$$

This cost function penalizes unnecessary node motions such that movements that only actuate the relevant nodes for task completion result in a lower cost. It is also closely related to the kinetic energy in the system.

S3.3 Control of robot arm using motion primitives

For the robot arm as shown in Fig. 2**d** in the main text, the goal is to let the two fingers reach a randomized target location. The initial configuration of the robot arm is randomized as follows: the whole arm rotates around the base structure, the lower arm rotates around the elbow, and the two fingers rotate around the wrist all for a certain degree randomly chosen from -15 to 15 degrees. The

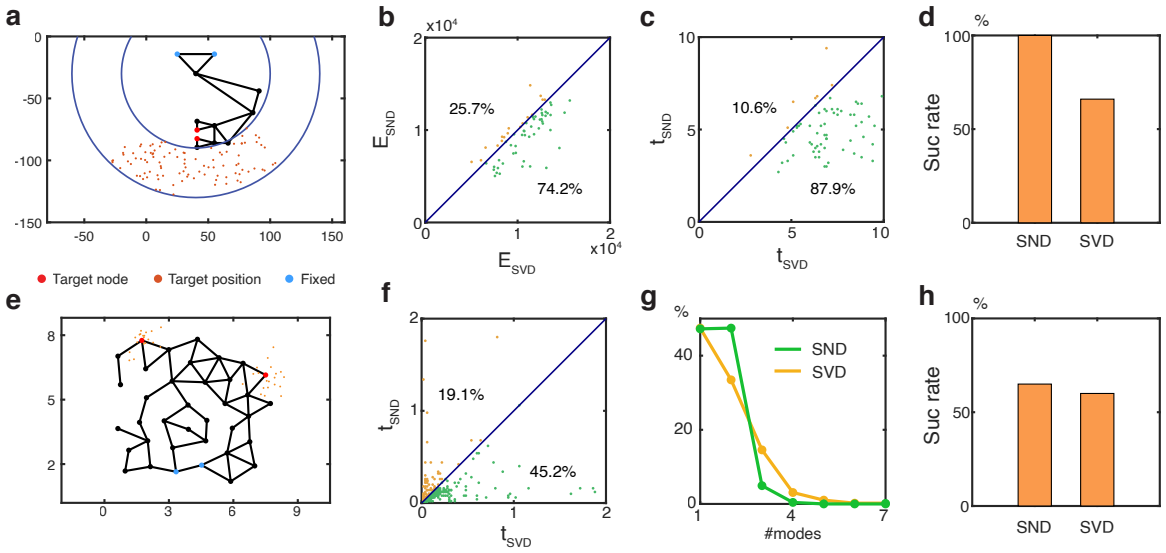

Figure S6: **Reaching control using motion primitives from SND and SVD modes. a–d**, Grasping control task of a robot arm. **a**, In this task the two target nodes are required to reach the target position within the two circles. **b**, Comparison of energy cost using motion primitives from SND and SVD among the successfully completed runs. **c**, Comparison of total simulation time. **d**, Comparison of success rate. **e-h**, Reaching control task for a random network. **e**, In each task, a node is required to reach the target position which follows a 2D normal distribution centered at the node's original position. **f**, Comparison of total simulation time using motion primitives from SND and SVD among successfully completed runs. **g**, Histogram of number of modes involved during the actuation process. **h**, Comparison of success rate.

target location is chosen randomly from a ring formed by two circles: between radius 85 and radius 110 from the base node (the node right below the two fixed nodes), and the ring spans from -45 degrees to 45 degrees from the vertical line. If the target location is too close to the current finger position (less than 50 from the lower finger), another random target location is drawn.

Mode selection The robot is then actuated based on the equations of motion in the previous section. To choose the mode to actuate at each time step, the modes are first normalized such that they all have the same infinite norm. If the two finger nodes marked as red in Fig. 2**d** are A and B, and the target is O, the optimal mode should be the one that minimizes the sum of the distances $d = |OA| + |OB|$. Since the robot arm has great flexibility to move compared to a more constrained random network, we let each mode to actuate for 0.2s and choose the mode which will result in the smallest d within the 0.2s.

Simulation This task is repeated 100 times with random initial configuration and target location, and for each configuration the task is repeated using motion primitives from both SND and SVD. Within each run, once a mode is chosen, it is actuated for 0.1s. The process stops when the two fingers are close to the target ($d < 7$). The actuation time of each mode is calculated as follows: the time is normalized from 0 (first actuated mode) to 1 (last actuated mode). Each mode thus has a sequence of actuation time, and the actuation time of a mode is the median of this time sequence.

Results Besides that the cascading activation of SND modes agree with the intuition of actuating an arm hierarchically, as shown in the main text, using SND as motion primitives also has the advantage

of costing less energy: it avoids unnecessary undulation of irrelevant nodes (See Movie S3). Among the runs where the fingers successfully reach the target, SND costs less energy than SVD in about 74% of the time (Fig. S6**b**). In addition, the motion is more coordinated and thus reaches the target easier (higher success rate among the 100 runs, Fig. S6**d**) and faster (88% of the time, Fig. S6**c**).

S3.4 Control of random network using motion primitives

The hierarchical actuation of the simple robotic arm might be obvious for human eyes. Therefore, we test the control on a random network using the motion primitives. We use the same network as in Section S2.3 where there are 14 DOF. For a randomly chosen node (target node), the task is to reach a target location nearby. The target location is randomly chosen from a 2D Gaussian distribution with the center being the location of the target node and width being 0.5. It is possible that when the target is too far from the node, the chosen node cannot reach the target however the modes are actuated.

The task is repeated for all the 35 nodes that are not fixed, and it is repeated 20 times for each node (20 random target location).

Mode selection The optimal mode i is chosen such that the distance between the target node and the target location reduces the fastest. In other words, the mode i^* is chosen such that

$$i^* = \underset{i\in\{1,\dots,14\}}{\operatorname{argmin}} |\vec{v}_{ij} \cdot \vec{d}|, \tag{S12}$$

where $\vec{v}_{ij}$ is the infinitesimal speed of target node j in ith mode, and $\vec{d}$ is the vector pointing from the target node to the target location.

Simulation The simulation is similar to the one for the robot arm above. After identifying the sequence of the modes actuated in each run, we also calculate the number of modes involved among the 14 modes.

Results Besides the plot for total energy as shown in the main text, Fig. S6**f** shows that the simulation using SND is also faster than that using SVD. Approximately 45% of the time SND is faster, compared to 19% the other way round (Fig. S6**f**). Note that due to the fact that the simulation is discretized and sometimes the target location can be close to the target nodes, the simulation time can be the same for SVD and SND.

Since SND modes are sparser and more spatially separated, the reaching tasks can involve less modes than in the case of SVD, as shown in the histogram in Fig. S6**g**. For very simple reaching tasks (target location very close to the current location of the target node), it is likely that actuating one mode is enough for both SVD and SND. For more complicated tasks, the reaching involves less modes with SND modes than that with SVD modes. In addition, it is also more likely that the reaching task completes successfully using SND as motion primitives (Fig. S6**h**).

S4 Network Control

S4.1 Spring network simulation

For the calculation of shear modulus, the rigid bonds are treated as springs with stiffness $k = 30$ and rest length $l_0 = 1.122$. There are in total N_x rows and N_y columns of nodes in the network. The nodes in the top row and the bottom row are fixed. At the beginning of the simulation, the nodes in the top row are displaced to the right by 8% of the width from top to bottom ($(N_y - 1)l_0 \times \sqrt{3}/2 \times 0.08$).

At each time step, the total force on each node is calculated as

$$\boldsymbol{F}_i = k \sum_{j \in G(i)} (l_{ij} - l_0)(\boldsymbol{x_j} - \boldsymbol{x_i})/l_{ij}, \tag{S13}$$

where $l_{ij} = ||\boldsymbol{r}_j - \boldsymbol{r}_i||$ and $G(i)$ is the set of nodes connected to node i by springs. The node is displaced by a small amount in the direction of the force (over-damped limit) plus a random noise:

$$x_{i,t+1} = x_{i,t} + F_i \Delta_t + \xi, \tag{S14}$$

where $\Delta_t = 0.0005$, and the small random uncorrelated noise ξ is uniformly distributed between -0.001 to $+0.001$ in both x and y direction. The noise is added to facilitate the equilibrium process. The process is repeated 20,000 times.

S4.2 Sequential tuning

The sequential tuning process shows the power of the hierarchical representation of floppy modes. By freezing the biggest modes, the system gets rigid faster than randomly adding links. The detailed procedure of MS protocol is as follows:

(a) Find the set of floppy modes using the SND algorithm.

(b) Find the mode with the largest s.

(c) Within this mode, find the node with the largest infinitesimal displacement.

(d) Connect this node to one of its neighbors. If there are multiple closest neighboring nodes available, randomly choose one.

(e) (For sequential tuning) Repeat steps (a) to (d) until the network is fully connected.

Fig. 3**g-h** in the main text only shows the results for 10 networks using the two methods. We apply this control protocol on 50 networks with random configurations (20% links connected), and repeat 10 times on each network. All the results and the median G values are shown in Fig. S7**a**.

In step (d), it is easy to identify the closest neighbors of a node since it is a regular triangular network. However, in the packing-derived network, one needs to set up the neighboring relationship beforehand, e.g. using a maximum neighboring distance. For the circular network example in Section S5.1, a fixed threshold is used to determine neighboring relationship.

S4.3 Multi-scale SND

Using multi-scale SND algorithm in the sequential tuning process can improve the efficiency even more. The step (a) in Section S4.2 is replaced by "Apply multi-scale SND algorithm to identify a set of modes in the network, with two nodes in the top row and the bottom row randomly selected to be spatially fixed." The resulting change of shear modulus suggests that this control protocol allows the network to achieve rigidity with even fewer number of links (Fig. S7**b**).

S4.4 Sampling frequency

The sequential tuning process in the main text requires re-calculating the rigidity matrix every time a link is added. In fact, we can reduce the sampling frequency. Once the modes have been sorted from the biggest to the smallest, links can be added based on the biggest q modes. Therefore, we add the following to the step (d) in the procedure above: Repeat for the largest q modes. The results are shown

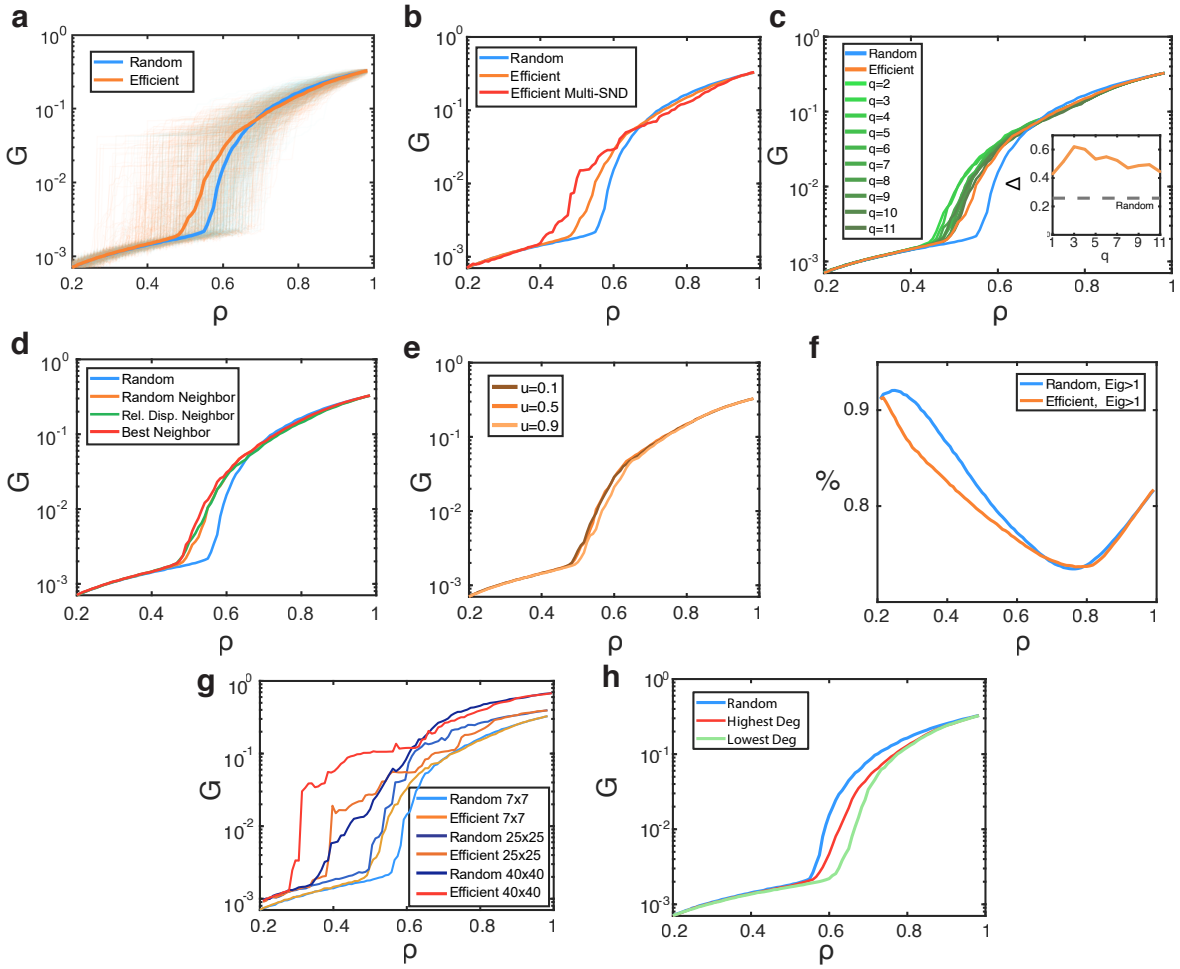

Figure S7: **Sequential tuning in triangular networks.** **a**, The change of shear modulus G in the sequential tuning process (similar to Fig. 3**g** in the main text), with 10 repeats from 50 random initial network configurations with 20% links connected. **b**, The multi-scale SND algorithm is added for comparison. **c**, A batch of q links (instead of one link) are added every time the mode decomposition is calculated, and the shear modulus is shown for each case. Inset shows the integration of G from $\rho = 0.35$ to 0.65 for different q. **d**, Comparison between the best neighbor approach, relative displacement neighbor approach, random neighbor approach, and the random case in the sequential tuning process. **e**, The effect of the parameter "u" in the sequential tuning process. **f**, The percentage of eigenvalues of $R^T R$ larger than 1 among all nonzero eigenvalues in the tuning process for both methods. **g**, Similar tuning process for a larger (25×25) network is added for comparison. **h**, Sequential tuning process with two heuristic approaches to add links: Always adding links to the highest-degree node or always adding links to the lowest-degree node.

in Fig. S7**c**. The tuning result from some intermediate q is better than the original $q = 1$ case. This might come from the fact that, since adding a link might not always kill a mode, the $q = 1$ case might get stuck in killing one particular mode and waste links during this process.

To quantify the difference between different sampling frequencies, the integration of G with respect

to the link density

$$\Delta(q) = \int G(\rho) d\rho \tag{S15}$$

from $\rho = 0.45$ to 0.65 is calculated for different q (Fig. S7**c** inset). For this 7×7 triangular network, the optimal sampling frequency is between 3 and 4.

S4.5 Neighbor selection in sequential tuning

In step (d) of the sequential tuning process, instead of adding a link to a randomly chosen neighbor, it is also possible to find the "best" neighbor to add a link between. To investigate the effect of the neighbor selection, we apply two approaches. (1) Relative displacement approach: the step (d) in the process is updated as "connect this node to the neighboring (unconnected) node which has the largest relative displacement when the mode is activated". In other words, assume that the current node c has infinitesimal speed $\boldsymbol{v}$ when the node is activated, find the closest (unconnected) neighboring node j such that $|(\boldsymbol{x_c} - \boldsymbol{x_j}) \cdot \boldsymbol{v}|$ is the largest. (2) Best neighbor approach: step (d) is updated as "calculate the shear modulus after each link from the node to its (unconnected) closest neighbors is added separately. Pick the link that leads to the largest increase in the shear modulus." Approach (1) has a very modest effect in terms of the control efficiency. Approach (2) takes significantly more time as it involves the spring network simulation until equilibrium. It does improve the tuning efficiency, but the effect is not significant either (Fig. S7**d**).

S4.6 Comparison with graph theory-based approaches

In the sequential tuning process, two graph theory-based approaches are implemented in comparison to the random protocol: (1) Always adding a link between the highest-degree node and one of its neighbors. (2) Always adding a link between the lowest-degree node and one of its neighbors. Neither of these two approaches are as good as the random protocol. This can be explained as follows: If links are always added to highest-degree nodes, the network is being strengthened at already highly-connected areas, which does not necessarily increase the stiffness overall. If links are always added to lowest-degree nodes, there are a lot of corner and edge nodes that are connected in the beginning, which do not help increasing the shear modulus of the network.

S4.7 Algorithm parameter u

We also examine the effect of the parameter u in the sequential tuning process. In the SND algorithm, u is a parameter used in a process similar to Markowitz's pivot selection (see equation (6) in [1]). In all the simulations above, we use $u = 0.5$. Changing u to be 0.1 or 0.9 does not change the tuning results significantly (Fig. S7**e**).

S4.8 Spectral analysis of the dynamical matrix

To further understand the underlying mechanism of efficient rigidification, we perform spectral analysis to the dynamical matrix

$$D = R^T \Lambda R, \tag{S16}$$

where Λ is a diagonal matrix with the diagonal entries corresponding to the stiffness of the springs. Since we assume equal stiffness in all the links, Λ is simply the identity matrix multiplied by a constant. The eigenvalues of the dynamic matrix D correspond to the frequencies of the modes. We repeat the sequential tuning in the main text, and calculate the distribution of eigenvalues of D each time after a link is added, for both our method and the random method.

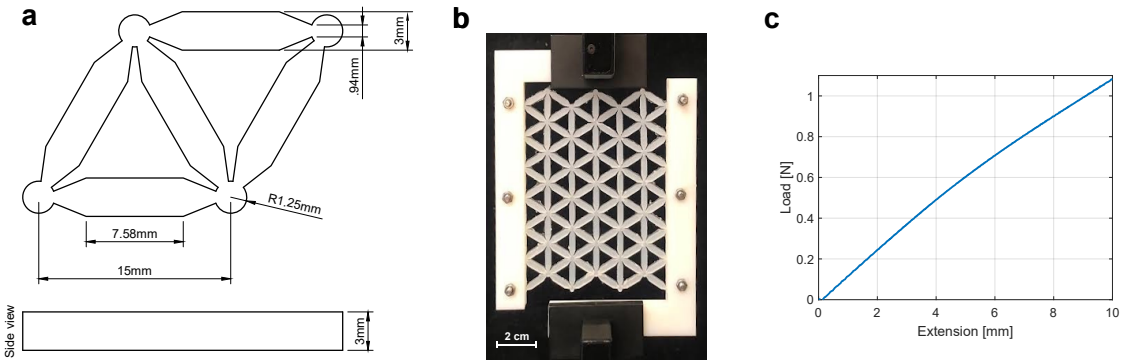

Figure S8: **Experimental network set-up and test. a**, Dimensions of a cell of the cast experimental network. **b**, Image of a cast network in the force measurement set-up. **c**, Load measurement raw data of a fully-connected 7×7 network for a maximal displacement of 10%.

When the link density is small, our method is mostly killing the largest zero mode, and this largest zero mode is more likely to become a low-frequency mode. Plotting the percentage of the mode frequencies above a threshold (one), we see that the MS protocol indeed builds networks with less high frequency modes (Fig. S7**f**). Therefore, a higher participation rate P is enforced in the MS protocol, thus increasing the shear modulus faster.

Note that the percentage of high-frequency modes (eigenvalues larger than 1) increases when ρ becomes larger. This comes from there being no floppy modes, and adding additional links makes the network overall more rigid with higher frequency modes.

S4.9 Validity of MS protocol

MS protocol, designed to rigidify the network, is only valid when the network is under-constrained. Note that this happens when the link density is below the jamming threshold. Moreover, the link density should also be below the rigidity transition threshold [9], where the jump happens in the random case.

To see how the system size affects the validity of MS protocol, we apply the similar control process as in Section S4.2 to a 25×25 and 40×40 network. The gap between the MS protocol and the random protocol is wider in the larger network, and the jump in shear modulus occurs at lower link density (Fig. S7**g**). This comes from the fact that in larger networks, the ratio of boundary nodes over bulk nodes is smaller, and the transition density is closer to the large network limit $\rho = 0.445$ [9].

S4.10 Experiments

Physical networks are cast using 3D printed molds and silicone rubber (Dragon Skin$^{\text{TM}}$ 30, Young's modulus 5.93×10^5 Pa). Each network edge has a length of 15 mm with a width of 3 mm that thins out towards the node (to 1 mm) to enable space between connecting edges to a node and to minimize rotational stiffness (see Fig. S8**a**). Each network is fully connected initially, and we cut connections sequentially if required by the experimental protocol. Shear forces are measured on an Instron 5566 with a 10N load cell by clamping the outermost edges of the network along its length with a 3D printed clamp and applying a displacement (at a rate of 1 mm/s) to one side until a displacement of 10% of the network length is reached. The occurring shear forces in this process are recorded as shown in Fig. S8**b** for a 7×7 network stretched by 10%. A small decrease from the initial slope is observed after 5% stretching which coincides with the out-of-plane motion of the network. Shear force data used for evaluation is

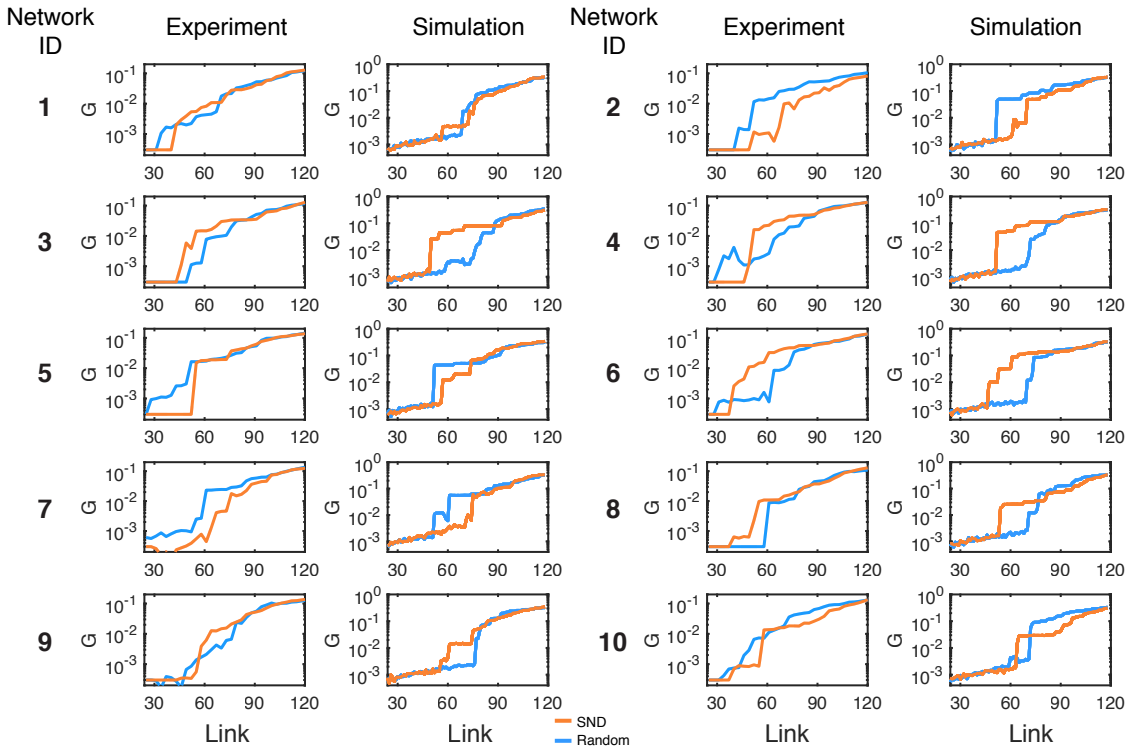

Figure S9: **The shear modulus of the 10 example networks with different links sequentially added/removed in both experiments and simulations.**

therefore extracted at 3% strain to avoid influence of out-of-plane effects. We tested 10 networks each with a different edge-removal sequences. The load curves of all 10 fully connected networks were checked to be nominal before experiments were conducted.

The raw data is shown in Fig. S9. Since the rotational energy plays a role at the small-stretch state, the shear modulus of the network in the experiment is nonzero even when there is no stretching involved. The biggest shear modulus we have observed when there should be zero stretch is in Network 8, and the shear modulus is around 0.01. Therefore, all the shear modulus below 0.01 are set to 0 in Fig. 3 of the main text.

The sequence of the links added in the experiment is shown in Fig. S10.

S5 Load prediction

S5.1 Generating jamming network

The standard jamming algorithm is used to create the spring network in the load prediction section [10–12]. Bi-dispersed disks grow until reaching jammed configuration in the confined square, and springs connect the center of disks if the two disks overlap. The network generated this way has higher heterogeneity compared to the regular triangular network.

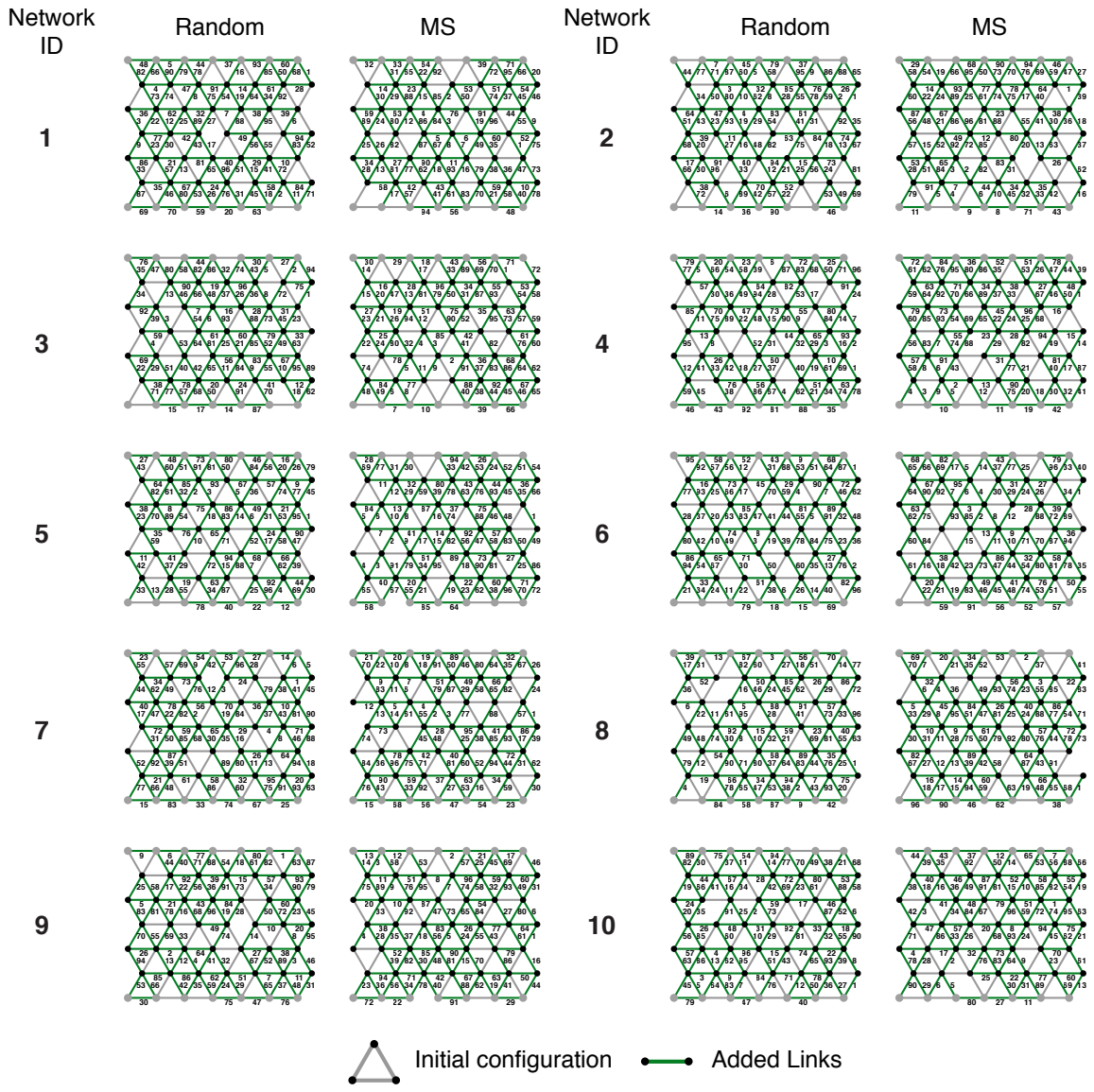

Figure S10: **The sequence of the links added to networks (green) in the experiment using the MS protocol or randomly.** The gray links form the initial network configuration.

S5.2 Load prediction

The key step in the load prediction is to find nodes that are involved in smaller modes in the SND mode representation, and it is less likely for the links between these nodes to bear loads. The intuition behind it is that, since the modes found by the SND algorithm are spatially separated and hierarchically represented, the nodes in smaller modes are more likely to be at free ends or have more flexibility (less interconnected) to accommodate the network deformation. Therefore, when the network is stretched, the links connecting those at free ends do not need to stretch, and the links connecting the flexible nodes can

rotate to accommodate the deformation.

Here we provide more details of the load prediction procedure.

Step 1: Mode decomposition

The first step is to identify how likely these nodes are in smaller modes (free ends or flexible area) or bigger modes (more interconnected area).

(a) Apply the SND method to find a set of floppy modes. Calculate the size of each mode s_j.

(b) Find out all the set of modes $\boldsymbol{B}_i$ that involve the movement of node i.

(c) Repeat (a) and (b) m times. Calculate the globality as

$$f_i = \frac{1}{m} \sum_{h=1}^{m} \min_{j \in \boldsymbol{B}_i^h} s_j^h,$$

where s_j^h is the size of the jth mode in the hth run, and $\boldsymbol{B}_i^h$ is the set of modes in the hth run involving node i ($h = 1, 2, ..., m$). If node i is not involved in any mode, f_i is set to be 0.

The calculation above assigns a quantity f_i for each node i, which characterizes how likely a node is involved in a larger or smaller mode. For example, a node at a free end is usually involved in a smaller mode, so the corresponding f_i is smaller. $m = 50$ in the circular network example.

Step 2: Find eligible nodes

The next step is to find the set of nodes that might be connected to links that bear loads. Setting a threshold t for the globality f, we define the nodes with $f_i > t$ (more likely to involve in larger modes) to be global nodes, and the nodes with $f_i < t$ (more likely to involve in smaller modes) as local nodes. For nodes that are not involved in any modes ($f_i = 0$), they are fixed nodes.

The eligible nodes are the union of global nodes, fixed nodes, and boundary nodes, as these nodes are believed to be more interconnected, the links among which are more likely to bear loads.

Step 3: Find potential load bearing path

We use the breadth-first search (BFS) from the boundary nodes (with the path only going through "fixed" and "global" nodes), and find a closest path to another boundary node. The prediction is that the links along the path are more likely to bear loads. The detailed procedure is as follows:

(a) Start from one of the boundary nodes and do a BFS. A descendent node has to satisfy three requirements: (1) It is within the set of eligible nodes. (2) It has not been visited. (3) It is connected to the current search node.

(b) At each layer in the search tree, mark the nodes as visited.

(c) Once one of the descendent nodes is a boundary node, stop the BFS search.

(d) The links along the shortest path between the two boundary nodes are predicted to bear higher loads. Remove "visited" markers generated in the current run.

(e) Repeat the process above until all the boundary nodes are visited. All the links not predicted to bear higher loads are predicted to bear less loads.

With the procedure above, all the links are classified to either bear more load or less load (more stretched or less stretched).

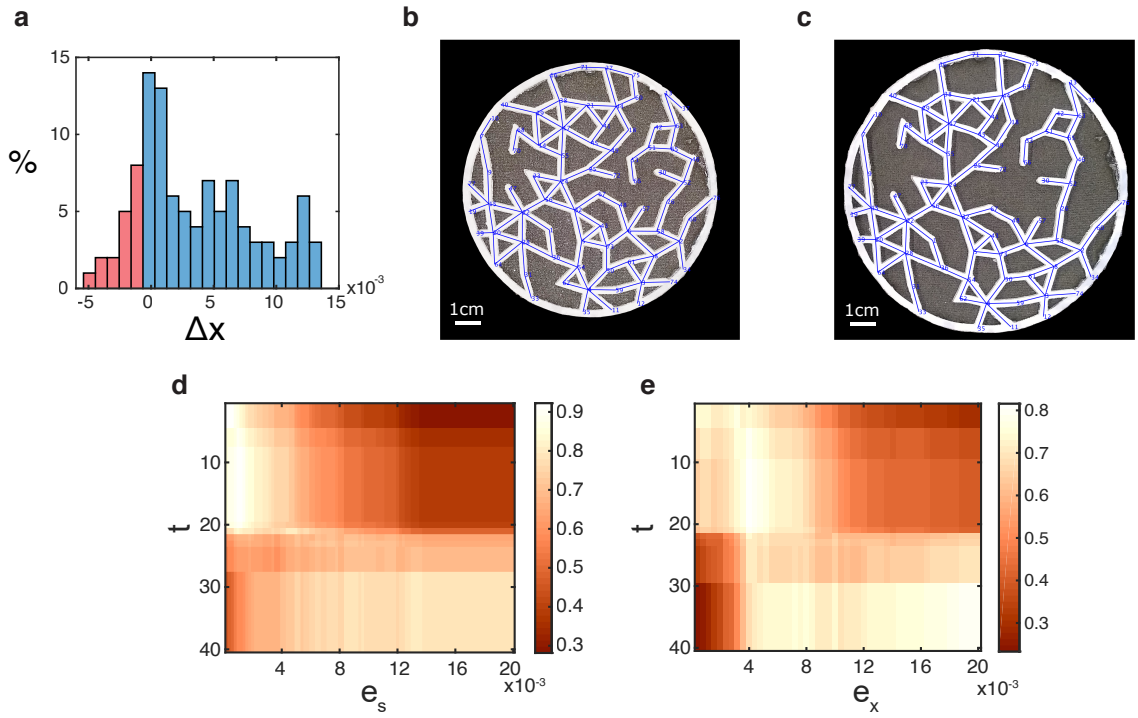

Figure S11: **Numerical simulation for load prediction.** **a**, The histogram of extension (or compression) of the springs in the numerical simulation indicates that most of the springs are stretched (blue) rather than compressed (red) under the boundary stretching. **b**, The network before stretching, with the nodes manually marked. **c**, The network after 11% stretching, with the nodes manually marked. **d**, The global fit for optimal prediction accuracy towards the simulation, with both threshold t and e_s varying in a range. Brighter color represents higher accuracy. **e**, A similar plot with the prediction accuracy for experiment with varying t and e_x.

S5.3 Simulation

In the numerical simulation, the network is treated as a spring network with spring stiffness $k = 30$, and the rest length l_0 is equal to the initial length after the jamming network is generated. The simulation is similar to the calculation of shear modulus: The boundary nodes are first displaced outwards from the center by 10%, and the network is equilibrated using the same integration steps as described in Section S4.1. Since the edge length is about 10 times smaller than that in the triangular network, the additional noise is reduced to -0.00005 to $+0.00005$ to facilitate equilibrium.

After the network is equilibrated, set a threshold e_s. Links with extension larger than e_s are classified as "more stretched", and those with extension smaller than e_s are classified as "less stretched". (The e_s shown in figures is rescaled by the diameter of the circle and so it is dimensionless.) This binary classification can be used to compare with the prediction from the infinitesimal approach.

Note that some links might be compressed rather than stretched under the global stretching. However, the percentage of the links compressed is small (Fig. S11**a** with the blue color showing the distribution of links stretching and the red color showing the distribution of links compressed), and so we only consider the absolute change in the length of the springs.

a

Simulation \ Prediction	Large e	Small e
Large e	60.2%	4.9%
Small e	1.9%	33.0%

b

Experiment \ Prediction	Large e	Small e
Large e	44.7%	1.0%
Small e	17.5%	36.9%

Table S1: **The prediction accuracy from the prediction towards the numerical simulation (a) and the experiment (b).** The accuracy is the sum of "true positive" rate (top left number) and "true negative" rate (bottom right number).

S5.4 Experiments

The circular network is cast using a 3D printed mold and silicone rubber (Dragon Skin$^{\text{TM}}$ 30, Young's modulus 5.93×10^5 Pa). The physical network diameter is 100 mm and the thickness of an edge is 2.5 mm. The outermost edge of the circular network is glued on a stretchable black spandex cloth which is uniformly stretched to reach an 11% size increase as shown in Fig. S11**b**. Network nodes are extracted and evaluated from image data. The final position of nodes are shown in Fig. S11**c**.

A similar threshold e_x (scaled by the diameter of the circle) is set up to classify the extension of the links into more stretched or less stretched.

S5.5 Prediction accuracy

Since we have the binary classification of links into either "more stretched" or "less stretched" in the prediction (from infinitesimal approach), simulation, and experiment, we can calculate the match percentage (prediction accuracy) between them. We define the "true positive" rate as the percentage of links stretched more in both prediction and simulation (or experiment), and "true negative" rate as the percentage of links stretched less in both prediction and simulation (or experiment). The prediction accuracy is the sum of the "true positive" rate and the "true negative" rate.

By varying the threshold of the binary classification, we might get different prediction accuracy. In Fig. 6**j** and **k** of the main text, we have already showed the prediction accuracy with $t = 12$ and varying threshold e_s and e_t in simulation and experiment. It is also possible to vary the threshold t and obtain an optimal accuracy in the two-dimensional parameter space. Fig. S11**d** shows the prediction accuracy between prediction and simulation with varying t and e_s, and Fig. S11**e** shows the accuracy between prediction and experiment with varying t and e_x. The highest prediction accuracy is 93.2% when t is from 9 to 20 and $e_s = 0.0018$ for the simulation, and 81.6% when t is between 9 and 20 and $e_x = 5$ for the experiment (see also Table S1).

S6 Video caption

Movie S1: Actuation of the modes in a 4×4 network. The four floppy modes identified by SND and SVD in Fig. 2 (main text) are actuated in this movie. Since some of the modes are only infinitesimal, they are actuated with small amplitudes.
Movie S2: Numerical simulation to find the equilibrium length of a network subject to uniform stretching. For the circular network, the boundary nodes are first displaced outwards from the center by 10%. The network is then equilibrated using the same integration procedures as described in Section S4.1. The first 100 integration steps are slowed down because the change in the spring length is faster in the beginning.

Movie S3: Pinching task performed by a robot arm using different motor primitives. The goal is to let the two fingers (red filled nodes) to reach the target position (red circle). The control process requires motor primitives, which can derive from either SND or SVD. With the motor primitives from SND the reaching is faster, more hierarchical (starting from the arm mode and ending with the finger modes), and more energy efficient (less unnecessary undulations at the end of the arm in the beginning).
Movie S4: Reaching task performed by a random network using different motor primitives. The goal is to let an arbitrary chosen node in the network to reach a randomly chosen target location near that node. With motor primitives from SND, the reaching is achieved faster and costs less energy.

References

[1] M. Khorramizadeh and N. Mahdavi-Amiri, "An efficient algorithm for sparse null space basis problem using abs methods," *Numer. Algorithms*, vol. 62, no. 3, pp. 469–485, 2013.

[2] J. Abaffy, C. G. Broyden, and E. Spedicato, "A class of direct methods for linear equations," *Numer. Math.*, vol. 45, pp. 361–376, 1984.

[3] J. Abaffy and E. Spedicato, *ABS projection algorithms: mathematical techniques for linear and nonlinear equations*. Upper Saddle River, NJ: Prentice-Hall, 1989.

[4] Z. Chen, N. Dang, and Y. Xue, "A general algorithm for underdetermined linear systems," in *Proceedings of the First International Conference on ABS Algorithms*, 1992.

[5] H. M. Markowitz, "The elimination form of the inverse and its application to linear programming," *Manage. Sci.*, vol. 3, no. 3, pp. 255–269, 1957.

[6] F. G. Woodhouse, H. Ronellenfitsch, and J. Dunkel, "Autonomous actuation of zero modes in mechanical networks far from equilibrium," *Phys. Rev. Lett.*, vol. 121, no. 17, p. 178001, 2018.

[7] J. Z. Kim, Z. Lu, S. H. Strogatz, and D. S. Bassett, "Conformational control of mechanical networks," *Nat. Phys.*, vol. 15, no. 7, pp. 714–720, 2019.

[8] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM J. Comput.*, vol. 1, no. 2, pp. 146–160, 1972.

[9] C. P. Broedersz, X. Mao, T. C. Lubensky, and F. C. MacKintosh, "Criticality and isostaticity in fibre networks," *Nat. Phys.*, vol. 7, no. 12, p. 983, 2011.

[10] M. Skoge, A. Donev, F. H. Stillinger, and S. Torquato, "Packing hyperspheres in high-dimensional Euclidean spaces," *Phys. Rev. E*, vol. 74, no. 4, p. 041127, 2006.

[11] W. G. Ellenbroek, Z. Zeravcic, W. van Saarloos, and M. van Hecke, "Non-affine response: Jammed packings vs. spring networks," *Europhys. Lett.*, vol. 87, no. 3, p. 34004, 2009.

[12] A. J. Liu and S. R. Nagel, "The jamming transition and the marginally jammed solid," *Annu. Rev. Condens. Matter Phys.*, vol. 1, no. 1, pp. 347–369, 2010.