Annals of Mathematics and Artificial Intelligence
https://doi.org/10.1007/s10472-023-09912-8

Optimizing doubly stochastic matrices for average consensus through swarm and evolutionary algorithms

Panos K. Syriopoulos[1] · **Konstantinos I. Chatzilygeroudis**[1] · **Nektarios G. Kalampalikis**[1] · **Michael N. Vrahatis**[1]

Accepted: 1 November 2023

Abstract
Doubly-stochastic matrices play a vital role in modern applications of complex networks such as tracking and decentralized state estimation, coordination and control of autonomous agents. A central theme in all of the above is consensus, that is, nodes reaching agreement about the value of an underlying variable (e.g. the state of the environment). Despite the fact that complex networks have been studied thoroughly, the communication graphs are usually described by symmetric matrices due to their advantageous theoretical properties. We do not yet have methods for optimizing generic doubly-stochastic matrices. In this paper, we propose a novel formulation and framework, EvoDSM, for achieving fast linear distributed averaging by: (a) optimizing the weights of a fixed graph topology, and (b) optimizing for the topology itself. We are concerned with graphs that can be described by positive doubly-stochastic matrices. Our method relies on swarm and evolutionary optimization algorithms and our experimental results and analysis showcase that our method (1) achieves comparable performance with traditional methods for symmetric graphs, (2) is applicable to non-symmetric network structures and edge weights, and (3) is scalable and can operate effectively with moderately large graphs without engineering overhead.

Keywords Average consensus · Distributed averaging · Evolutionary algorithms · Sparse matrix identification · Particle swarm optimization

Mathematics Subject Classification (2010) 93D50 · 68W50 · 68T20 · 62M45

✉ Panos K. Syriopoulos
p.syriopoulos@math.upatras.gr

Konstantinos I. Chatzilygeroudis
costashatz@upatras.gr

Nektarios G. Kalampalikis
nkalamp@math.upatras.gr

Michael N. Vrahatis
vrahatis@math.upatras.gr

[1] Computational Intelligence Laboratory (CILab), Department of Mathematics, University of Patras, GR-26110 Patras, Greece

Published online: 08 June 2024

1 Introduction

A complex network is a set interacting agents connected through a graph structure. The study of complex networks involves analyzing the dynamics between connected agents as well as the dynamics of the graph structure itself. The combination of these dynamics produces emergent behaviors even for seemingly simple interactions (we refer the reader to [1] for more details about the major concepts). Technological advancements throughout the years have called for a better understanding of complex networks. With the introduction of the multiprocessor, intricately connected computing agents created a need for dynamic load balancing methods through local exchanges of data (cf. [2]). Now, with recent advancements in wireless communications, there is a greater endeavor in automating, coordinating and controlling systems of interacting robots in a distributed fashion. Applications span from coordination of mobile agents [3–5], object tracking in sensor networks [6] and network connectivity maintenance [4], estimation and control [7] and optimization [8, 9]. The study of social networks is also an active area of research [10].

A basic but integral part of distributed systems is *consensus*. In this paper, we are interested in the special case of *distributed averaging* [11, 12]. In this instance the nodes have to agree on the average of their initial measurements by communicating through the network. A tutorial for applications in control can be found in [13] and a recent distributed consensus algorithm in [14]. It is well-known that the speed with which a system converges to consensus depends upon the eigenvalues of the underlying network weight matrix. This is the case for both static models (as we formulate in Section 2) and dynamic models [14]. However, optimizing network weights for speed is a spectral radius minimization problem [15], which admits a convex formulation for symmetric weight matrices only. In this paper we show that it is viable to use evolutionary algorithms to find near optimal non-symmetric weight matrices. Our experiments show, optimizing non-symmetric weight matrices has merits, the asymptotic speed of convergence can be considerably faster with asymmetric network weights. Moreover our proposed framework allows for applications in doubly-stochastic matrix construction and (graph) topology discovery.

We consider discrete time distributed averaging with linear update rules. In each communication round, each node updates its estimate by taking a weighted sum of the estimates of its neighbors. We are concerned with

(a) the problem of the *speed of convergence to the average* and
(b) the problem of *optimizing the network itself*.

In other words, for problem (a) we want to minimize the number of communication rounds required for all the nodes to approximate the average within an error bound. The asymptotic speed of convergence is bounded above by the second largest eigenvalue in magnitude. We optimize the network weights for speed while respecting the network connectivity constraints. For problem (b), given a target for the asymptotic speed of convergence, we aim to find a network structure that is as *sparse* as possible. We tackle these problems using the *Unified Particle Swarm Optimization* (UPSO) [16–19], and *Covariance Matrix Adaptation Evolutionary Strategy* (CMA-ES) [20].

Evolutionary algorithms are (i) easy to parallelize, (ii) can handle non-differentiable and even discontinuous objective functions, and (iii) it has been showcased that they can find creative solutions to hard problems [21]. As a result, we are interested to study how these algorithms perform in the aforementioned problems (a) and (b). We propose a novel framework, called EVolutionary Optimization of Doubly Stochastic Matrices (EvoDSM), that is

able to optimize arbitrary positive doubly stochastic matrices and provide practical solutions to problems (a) and (b).

Our contributions can be summarized as follows:

(1) Practical framework for optimizing arbitrary positive doubly stochastic matrices;
(2) An iterative normalization scheme that maps arbitrary matrices to doubly stochastic ones for making the optimization tractable;
(3) Extensive experimental validation of the proposed method in many different graphs of different sizes.

The structure of the paper is as follows. Section 2 contains the background material and relevant approaches in the literature. In Section 3 we present our framework and the algorithms used in detail. Section 4 contains experiments and analysis and Section 5 provides a conclusion and directions for future work.

2 Background and related work

Consider a directed graph $G(V, E)$ where $V = \{1, 2, \ldots, n\}$ is the vertex set and E is the ordered set containing the edges $E = \{(i, j) : \text{node } i \text{ listens to node } j\}$. The set of weights that nodes give to their neighbors can be represented by a stochastic matrix $\boldsymbol{W}$ with elements $\boldsymbol{W}_{i,j}$ such that:

$$\boldsymbol{W}_{i,j} = \begin{cases} \boldsymbol{W}_{i,j} > 0, & \text{if} \quad \{i, j\} \in E, \\ 0, & \text{if} \quad \{i, j\} \notin E. \end{cases} \tag{1}$$

Denote by $\boldsymbol{x}(0)$ the vector of initial values held by the nodes such that $\boldsymbol{x}_i(0)$ is the value held by node i. The linear update rule (*consensus model of DeGroot* [22]) is given by:

$$x(t+1) = \boldsymbol{W}\boldsymbol{x}(t) = \boldsymbol{W}^{t+1}\boldsymbol{x}(0). \tag{2}$$

The *consensus vector* denoted by $\boldsymbol{x}_c$ is given by the following limit, provided it exists:

$$\boldsymbol{x}_c = \lim_{t\to\infty} \boldsymbol{x}(t) = \lim_{t\to\infty} \boldsymbol{W}^t\boldsymbol{x}(0). \tag{3}$$

It can be seen that existence and uniqueness of the limit above is equivalent to existence and uniqueness of a left eigenvector $\boldsymbol{\pi}$ of $\boldsymbol{W}$ with eigenvalue equal to one:

$$\boldsymbol{\pi}\boldsymbol{x}(t+1) = \boldsymbol{\pi}(\boldsymbol{W}x(t)) = (\boldsymbol{\pi}\boldsymbol{W})\boldsymbol{x}(t) = \cdots = \boldsymbol{\pi}\boldsymbol{x}(0). \tag{4}$$

In terms of the graph induced by $\boldsymbol{W}$, it is well known that necessary and sufficient conditions for the existence of the limit in (3) are equivalent to $G(V, E)$ being strongly connected and aperiodic. Aperiodicity is usually enforced by requiring that the diagonal elements of $\boldsymbol{W}$ are bounded below by a small positive number. Additionally, for average consensus, the weight structure needs to be *balanced* in the sense that $\sum_{j=1}^{n} \boldsymbol{W}_{i,j} = \sum_{j=1}^{n} \boldsymbol{W}_{j,i}$ (see for example [13]). This is equivalent to $\boldsymbol{W}$ being doubly-stochastic, i.e. $\sum_{j}^{n} \boldsymbol{W}_{i,j} = \sum_{j}^{n} \boldsymbol{W}_{j,i} = 1$ for all $i \in V$.

Provided the necessary and sufficient conditions for consensus are satisfied, it can be seen that the speed of convergence of (2) depends on the second largest eigenvalue in magnitude.

Specifically, assuming the matrix $\boldsymbol{W}$ is generic (i.e., non-singular), a spectral decomposition reveals that:

$$\boldsymbol{W} = \quad \boldsymbol{P}\boldsymbol{D}\boldsymbol{P}^{-1} = \quad \lambda_1\boldsymbol{M}_1 + \lambda_2\boldsymbol{M}_2 + \cdots + \lambda_n\boldsymbol{M}_n$$

$$\boldsymbol{W}^t = \quad \boldsymbol{P}\boldsymbol{D}^t\boldsymbol{P}^{-1} = \quad \lambda_1^t\boldsymbol{M}_1 + \lambda_2^t\boldsymbol{M}_2 + \cdots + \lambda_n^t\boldsymbol{M}_n$$

with $\boldsymbol{D}$ the diagonal matrix $\boldsymbol{D} = \text{diag}\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$, $\boldsymbol{P} = [f_1, f_2, \ldots, f_n]^\top$ where f_i's are the right eigenvectors of $\boldsymbol{W}$ and $\boldsymbol{P}^{-1} = [\pi_1, \pi_2, \ldots, \pi_n]$ where π_i's are the left eigenvectors of $\boldsymbol{W}$. Ordering the eigenvalues in decreasing magnitudes, and since $\lambda_1 = 1$ is the unique dominant eigenvalue, it is clear that λ_2 has the slowest decay as the number of iterations t increases. More specifically, by linear independence we have:

$$f_i\pi_j = \begin{cases} 0, & \text{if} \quad i \neq j, \\ 1, & \text{if} \quad i = j. \end{cases}$$

Then, the matrices M_k for $i = 1, 2, \ldots, n$ can be defined as:

$$M_k = f_k\pi_k^T = \begin{bmatrix} f_k(1)\,\pi_k(1) & \ldots & f_k(1)\,\pi_k(n) \\ \vdots & \ddots & \vdots \\ f_k(n)\,\pi_k(1) & \ldots & f_k(n)\,\pi_k(n) \end{bmatrix}.$$

The result follows by checking that:

$$M_i Mj = \begin{cases} 0, & \text{if} \quad i \neq j, \\ M_i, & \text{if} \quad i = j. \end{cases}$$

Therefore the problem of fast consensus relates to the problem of "second largest eigenvalue magnitude minimization" (SLEM) which solves the *Fastest Mixing Markov Chain* (FMMC) problem. We refer the reader to [23] for more details.

Boyd *et al.* [15, 24] formulated SLEM minimization for symmetric matrices as a *Semi-Definite Program* (SDP). More precisely, they formulated the following problems:

$$\begin{aligned} &\min\ \varrho(\boldsymbol{W} - \boldsymbol{1}\boldsymbol{1}^\top n^{-1}), \\ &\text{s.t.}\quad \boldsymbol{W} \in B, \\ &\qquad \boldsymbol{1}^\top\boldsymbol{W} = \boldsymbol{1}^\top, \\ &\qquad \boldsymbol{W}\boldsymbol{1} = \boldsymbol{1}. \end{aligned} \qquad \text{and} \qquad \begin{aligned} &\min\ \left\| \boldsymbol{W} - \boldsymbol{1}\boldsymbol{1}^\top n^{-1} \right\|, \\ &\text{s.t.}\quad \boldsymbol{W} \in B, \\ &\qquad \boldsymbol{1}^\top\boldsymbol{W} = \boldsymbol{1}^\top, \\ &\qquad \boldsymbol{W}\boldsymbol{1} = \boldsymbol{1}. \end{aligned} \tag{5}$$

The matrix $\boldsymbol{1}\boldsymbol{1}^\top n^{-1}$ is known as the *averaging matrix* and corresponds to a complete graph with all weights equal to $1/n$. The first problem minimizes the spectral radius (i.e., the largest absolute value of the eigenvalues of the matrix) and is generally hard to solve because it is non-convex and not Lipschitz continuous [25]. The second problem minimizes the spectral norm, that is, $\|\boldsymbol{W}\|$ is the largest singular value of $\boldsymbol{W}$. If $\boldsymbol{W}$ is constrained to be symmetric, then the two problems coincide. The set B represents the communication restrictions of the network and it corresponds to (1). To get an intuitive understanding of these formulations, one can observe that the complete graph with equal weights ($\boldsymbol{1}\boldsymbol{1}^\top n^{-1}$) yields the fastest possible network. It corresponds to full connectivity (all nodes communicate with all other nodes), and it converges to the average in one iteration. Furthermore, subtracting the averaging matrix

from $\boldsymbol{W}$ deflates the largest eigenvalue (which equals 1) to zero. We would like to attempt the spectral radius minimization (non-convex) problem with evolutionary algorithms.

3 Optimizing doubly stochastic matrices with swarm and evolutionary algorithms

In this section, we outline our novel framework for optimizing arbitrary positive doubly stochastic matrices. We call this framework *Evolutionary Optimization of Doubly Stochastic Matrices* (EvoDSM). EvoDSM is capable of optimizing both the weights of a fixed topology graph as well as the topology itself, that is, identify which weights should be bigger than zero.

3.1 Optimization of real-valued objective functions

We consider an optimization problem as follows:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, J(\theta) \tag{6}$$

where $\boldsymbol{\theta} \in \mathbb{R}^m$, m being the dimensions of the search space and $J(\cdot)$ is the objective function.

In order to practically solve any optimization problem we need to choose:

(a) The optimization algorithm;

(b) The parameterization of the input space $\boldsymbol{\theta}$;

(c) The objective function $J(\cdot)$.

3.2 Swarm and evolutionary optimization algorithms

In this work, we utilize evolutionary algorithms for solving the optimization problem. Evolutionary algorithms are versatile optimizers that can even optimize discontinuous and highly non-linear functions [21, 26]. We will experiment with two of the most widely used evolutionary algorithms: (a) *Particle Swarm Optimization* (PSO) [18, 27] and (b) *Covariance Matrix Adaptation Evolution Strategy* (CMA-ES) [28]. We chose PSO as a representative of simple population-based methods and because it has many common features with pure genetic algorithms (most importantly, little to no overhead per iteration and big population sizes), while CMA-ES is one of the most effective algorithms that falls in the evolution strategies family [29, 30].

Particle Swarm Optimization (PSO) algorithms [18, 27] are stochastic optimization methods inspired by the aggregating behaviors of populations. A swarm of particles that communicate through a graph are randomly positioned in the search space. They perform evaluations of the objective function at their current positions respectively and decide on favorable directions of movement based on memory of the best positions found in the process. The two main strategies are local PSO (PSO-LS) and global PSO (PSO-GS). In the local strategy, each particle has memory of the its best position ever visited, and the single best position ever visited by its neighbors. In the global strategy particles maintain memory of their personal best position and the best position ever found by the swarm in aggregate. The interested reader can find a MATLAB code for PSO in [27].

A predetermined set of N particles are initialized at random positions in the search space. The position of particle i at initialization (first iteration, $t = 0$) is denoted by $\boldsymbol{\theta}_i(0) \in \mathbb{R}^m$. Additionally, each particle is provided with a random velocity vector denoted $\boldsymbol{v}_i(0) \in \mathbb{R}^m$. At each iteration $t + 1$, particles update their positions with the following equations:

PSO Local Strategy (PSO-LS)

$$\begin{aligned} \boldsymbol{v}_i(t+1) &= \chi\left[\boldsymbol{v}_i(t) + c_1 r_1\left(\boldsymbol{\theta}_i^b(t) - \boldsymbol{\theta}_i(t)\right) + c_2 r_2\left(\boldsymbol{\theta}_i^{lb}(t) - \boldsymbol{\theta}_i(t)\right)\right], \\ \boldsymbol{\theta}_i(t+1) &= \boldsymbol{\theta}_i(t) + \boldsymbol{v}_i(t+1). \end{aligned} \tag{7}$$

PSO Global Strategy (PSO-GS)

$$\begin{aligned} \boldsymbol{v}_i(t+1) &= \chi\left[\boldsymbol{v}_i(t) + c_1 r_1\left(\boldsymbol{\theta}_i^b(t) - \boldsymbol{\theta}_i(t)\right) + c_2 r_2\left(\boldsymbol{\theta}^b(t) - \boldsymbol{\theta}_i(t)\right)\right], \\ \boldsymbol{\theta}_i(t+1) &= \boldsymbol{\theta}_i(t) + \boldsymbol{v}_i(t+1). \end{aligned} \tag{8}$$

where $\boldsymbol{\theta}_i^b(t)$ denotes the best position visited by particle i from initialization and up to time t, and similarly, $\boldsymbol{\theta}_i^{lb}(t)$ and $\boldsymbol{\theta}^b(t)$ denote the best position ever found by the neighbors of i and the swarm in aggregate respectively. Scalars c_1, c_2 are user defined parameters and $r_1, r_2 \in [0, 1]$ are independent and randomly generated numbers[1]. Scalar χ is a user defined parameter similar to what learning rate is in gradient descent.

The *Unified Particle Swarm Optimizer* (UPSO) is an algorithm that combines the behaviors of the local and global PSO strategies. The update equations are given by:

UPSO method

$$\begin{aligned} \boldsymbol{g}_i(t+1) &= \chi\left[\boldsymbol{v}_i(t) + c_1 r_1\left(\boldsymbol{\theta}_i^b(t) - \boldsymbol{\theta}_i(t)\right) + c_2 r_2\left(\boldsymbol{\theta}^b(t) - \boldsymbol{\theta}_i(t)\right)\right], \\ \boldsymbol{l}_i(t+1) &= \chi\left[\boldsymbol{v}_i(t) + c_1' r_1'\left(\boldsymbol{\theta}_i^b(t) - \boldsymbol{\theta}_i(t)\right) + c_2' r_2'\left(\boldsymbol{\theta}_i^{lb}(t) - \boldsymbol{\theta}_i(t)\right)\right], \\ \boldsymbol{v}_i(t+1) &= u\,\boldsymbol{g}_i(t+1) + (1-u)\,\boldsymbol{l}_i(t+1), \\ \boldsymbol{\theta}_i(t+1) &= \boldsymbol{\theta}_i(t) + \boldsymbol{v}_i(t+1), \end{aligned} \tag{9}$$

where $u \in [0, 1]$ is a user defined parameter. Notice that if $u = 0$, UPSO coincides with the local strategy, PSO-LS, whereas, if $u = 1$, UPSO coincides with PSO-GS. In that sense, UPSO gives the "best of both worlds" by allowing the user to achieve superior performance with the fine-tuning of a single parameter.

Notice that in all strategies the velocity vectors $\boldsymbol{v}_i(t)$ are updated to point towards the best known positions. Intuitively, PSO-LS's velocity vectors "simulate" the gradient of the objective function locally, while PSO-GS slowly directs the particles towards the best position found globally. The particle count and the communication graph of the particles directly affect their performance. In this work, we use the UPSO method. The interested reader can find a MATLAB code for UPSO in [18].

3.2.1 Covariance matrix adaptation evolutionary strategies

Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [28] is a stochastic, derivative-free method for numerical optimization of non-linear non-convex continuous optimization

[1] The scalar product has the straight-forward interpretation of taking a step towards the personal best position ($\theta_i^b(t)$) and then taking a step towards the local/global best position (($\theta^b(t)$ or ($\theta^{lb}(t)$)). However, r_1 and r_2 are usually defined as independent and randomly generated vectors that multiply the corresponding factors component-wise (i.e. with the Hadamard product).

problems that has been successfully used in many real-world settings [31, 32]. CMA-ES models a population of points as a multivariate normal distribution and performs the following steps at each generation t (for more details see [32]):

(a) Sample λ new offspring according to a multi-variate Gaussian distribution of mean $\boldsymbol{m}_t$ and covariance $\sigma_t^2 \boldsymbol{C}_t$, that is, $\boldsymbol{\theta}_i \sim \mathcal{N}(\boldsymbol{m}_t, \sigma_t^2 \boldsymbol{C}_t)$ for $i = 1, 2, \ldots, \lambda$;
(b) Rank the λ sampled candidates based on their performance and select the fittest μ individuals with $\mu \leqslant \lambda$;
(c) To reflect the distribution of the μ best candidates, compute $\boldsymbol{m}_{t+1}$ by averaging the μ individuals $\boldsymbol{m}_{t+1} = \frac{1}{\mu} \sum_{i=1}^{\mu} \boldsymbol{\theta}_i$, and $\sigma_{t+1}^2 \boldsymbol{C}_{t+1}$.

3.3 Efficiently encoding doubly stochastic matrices

Now we turn our attention to the parameterization of the input space, $\boldsymbol{\theta}$. First, we need to recall that we want to optimize over square weight matrices of the form $\boldsymbol{W} \in \mathbb{R}^{n \times n}$, where n is the number of nodes in the graph. We additionally assume that $\boldsymbol{W} > 0$ and that $\boldsymbol{W}$ is a doubly stochastic matrix. The most obvious and naíve way would be to unfold the matrix into a vector $\boldsymbol{\theta} \in \mathbb{R}^{n^2}$ and create an objective function J that penalizes elements of the matrix that do not satisfy the constraints. While this is certainly possible, the optimizer will spend a lot of time to find matrices that satisfy the constraints and it will be difficult to make an objective function that can incorporate both the constraints and the actual objective that we want to optimize.

For that reason, for parameterization of the input space we propose using only the non-zero values of the matrix $\boldsymbol{W}$ without any constraints (i.e., they can take negative values and do not respect the doubly stochasticity) and employ an iterative normalization scheme that transforms the input encoding to a valid doubly stochastic matrix. So, overall we first take the input vector $\boldsymbol{\theta}$ and we transform it into a non-negative matrix $\boldsymbol{M}$. Afterwards, we perform iterative normalization on $\boldsymbol{M}$ and we obtain a matrix $\boldsymbol{W}$ that is doubly stochastic by construction. In this manner, the optimizer only needs to handle the optimization of our actual objective and not how to create doubly stochastic matrices.

The iterative normalization scheme we use is related to the Sinkhorn-Knopp algorithm [33]. Given a non-negative matrix $\boldsymbol{M}$, the Sinkhorn-Knopp algorithm is concerned with finding diagonal matrices $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ such that $\boldsymbol{M}_1 \boldsymbol{M} \boldsymbol{M}_2$ is doubly stochastic. The condition for existence and uniqueness of $\boldsymbol{M}_1, \boldsymbol{M}_2$ is that $\boldsymbol{M}$ has total support. However, a process which normalizes iteratively the sums of the rows and columns of $\boldsymbol{M}$ converges to a doubly-stochastic matrix and only requires support. Trivially, by enforcing that elements of the main diagonal of $\boldsymbol{M}$ be bounded below by some small positive number (to ensure aperiodicity), then $\boldsymbol{M}$ immediately satisfies the requirement of iterative normalization. Thus, iterative normalization serves the purpose of a projection from $\mathrm{R}^{n \times n}$ to the convex set of positive doubly-stochastic matrices.

Algorithm 1 describes the iterative normalization scheme. The argument "iter" controls how many times to normalize each row and each column. Lines 9-13 check that the resulting matrix is doubly-stochastic within a predefined tolerance level. For short, we denote the normalization scheme as $N(\cdot)$. It should be seen that the sparsity pattern of $\boldsymbol{W}$ remains unchanged, i.e. if $\boldsymbol{W}_{i,j} = 0$ for some $i, j \in \{1, 2, \ldots, n\}$, and $\boldsymbol{W}' = N(\boldsymbol{W})$, then $\boldsymbol{W}'_{i,j} = 0$ as well. This crucial property enables the use of iterative normalization in the context of graph topology discovery. Finally, it is shown that this iterative normalization converges in linear time [33].

Algorithm 1 Iterative Normalization I

```
function NORMALIZE(W, iter)
    dim ← the dimension of W, i.e. n
    for i = 0, 1, ..., 2 iter do
        for j = 0, 1, ..., dim do
            W_j ← W_j / (Σ_{k=1}^{dim} W_{j,k})
        end for
        W ← W^⊤
    end for
    if W is doubly-stochastic then
        return W
    else
        Normalize(W)
    end if
end function
```

3.4 Objective function for convergence speed

Once we know how to parameterize our matrices in an effective manner, we need to see how to define the objective function for solving the problems described in the introduction. We will first consider the problem of consensus convergence speed. As discussed, this problem is equivalent to the *second largest eigenvalue magnitude* (SLEM) problem.

To solve this problem, we will minimize SLEM directly. Although, instead of deflating the dominant eigenvalue like in (5), we calculate the two dominant eigenvalues of $\boldsymbol{W}$ directly. The reason for this is important: there is a chance that deleting an edge of $G(V, E)$ (i.e. updating $\boldsymbol{W}_{i,j}$ to zero for some i, j) disconnects the graph in two connected components. That would result in higher geometric multiplicity of the dominant eigenvalue, which is to be penalized. Algorithm 2 describes our proposed objective function.

Algorithm 2 SLEM Minimization Objective

```
function J_SLEM(θ)
    W ← the matrix associated of vector θ
    W' ← N(W)
    λ1, λ2 ← magnitudes of the dominant eigenvalues of W'
    if λ1 = λ2 then
        return ∞
    else
        return λ2
    end if
end function
```

Since $\boldsymbol{W}'$ is always a doubly-stochastic matrix, we always have $\lambda_1 = 1$. As shown in Section 2, $\lambda_1 = \lambda_2$ violates the uniqueness of the left eigenvector of $\boldsymbol{W}$, and therefore, $J_{\text{SLEM}}(\boldsymbol{\theta})$ returns infinite penalty. Otherwise, $J_{\text{SLEM}}(\boldsymbol{\theta})$ returns λ_2, which is the quantity to be minimized. Thus, J_{SLEM} is a well-defined objective function for the problem of interest, penalizing both disconnected graphs and aperiodic graphs.

3.5 Objective function for graph sparsification

We will now consider the problem of optimizing the topology of the network itself to achieve a specific convergence speed. As discussed, we propose to tackle this problem as the problem of finding a maximally sparse weight matrix while achieving a specified convergence speed. We would like our sparse matrix to attain a pre-specified SLEM target, denoted by λ_t. That is, we want $\lambda_2 = \lambda_t$. To this end we define a deviation coefficient $D(\lambda_2) = \max\{\frac{\lambda_2}{\lambda_t}, \frac{\lambda_t}{\lambda_2}\}$. Notice that $D(\lambda_2) > 1$ if $\lambda_2 \leqslant \lambda_t$, while $D(\lambda_2) = 1$ when $\lambda_2 = \lambda_t$. One should further normalize D by its maximum value given by $n^2 - n = n(n-1)$.

Next we quantify the degree of sparsity of the matrix $\boldsymbol{W}$. A straight forward approach would be to count the number of non-zero entries. To further facilitate the evolutionary optimizers, we find a relaxation of the above that penalizes small entries. This is done with the help of the adjacency matrix of $G(V, E)$, denoted by $\boldsymbol{A}$ and satisfying:

$$\boldsymbol{A}_{i,j} = \begin{cases} 1, & \text{if } (i,j) \in E, \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

The $L_{1,1}$ norm of $\boldsymbol{A}$, defined as $\|\boldsymbol{A}\|_{1,1} = \sum_{i,j} \mid A_{i,j} \mid$ is a natural count of the edges of $G(V, E)$. The relaxed measure of sparsity we are interested in, denoted by S, is given by $S = \|\boldsymbol{A} - \boldsymbol{W}\|_{1,1}$. With this definition, a small entry of $\boldsymbol{W}$ contributes more to the measure of sparsity, S, than a larger entry. This will help the evolutionary algorithms determine favorable directions of improvement. Observe that the two quantities S and D are competing. Minimizing S alone would result to the matrix of all zeros, but at the same time induce large deviation from the target SLEM. We induce a trade-off between S and D by taking the product SD. The trade-off provides enough leeway for the evolutionary algorithms to deviate from λ_t while searching for sparse matrices. We can tighten the search by raising the deviation term to a power, e.g. SD^5, which results to sparse matrices with smaller deviations from λ_t. Normalization of the deviation term D is also important for controlling the error, i.e. if W is a large matrix, deleting multiple edges at once may induce a favorable trade-off no matter how large the deviation from the target. Therefore, for the remained of the paper, we consider $D(\lambda_2)$ to be normalized by its maximum possible value $n(n-1)$.

A final remark concerns the robust calculation of the objective function. With the definition of $\boldsymbol{A}$ above (10), small entries of $\boldsymbol{W}$ will result in unit entries of $\boldsymbol{A}$. Adding a threshold in (10), will result in a miss-match between the calculated eigenvalues λ_1, λ_2 and graphs related to adjacency matrix $\boldsymbol{A}$. The latter might be disconnected, while at the same time we have $\lambda_1 \neq \lambda_2$ due to the effect of unaccounted small values. This issue is easily avoided by filtering small entries of $\boldsymbol{W}$ before constructing the adjacency matrix $\boldsymbol{A}$.

Algorithm 3 Filtering small values

```
1: function FILTER(W, tol)
2:     for every entry of W do
3:         if W_{i,j} ⩽ tol then
4:             W_{i,j} = 0
5:         end if
6:     end for
7: end function
```

Algorithm 4 Sparsification Objective

```
1: function J_SPARSIFY(θ)
2:     W ← the matrix associated to vector θ
3:     W ← FILTER(W, ϵ)
4:     W ← N(W)
5:     λ1, λ2 ← magnitudes of the dominant eigenvalues of W
6:     A ← adjacency matrix associated with W
7:     D ← calculate deviation from target
8:     S ← calculate sparsity measure
9:     if λ1 = λ2 then
10:        return ∞
11:    else
12:        return SD^5
13:    end if
14: end function
```

Algorithm 3 shows the filtering procedure, while Algorithm 4 defines the objective function $J_{\text{SPARSIFY}}(\cdot)$ for the task. Similarly to the previous case, we calculate both λ_1 and λ_2 to eliminate reducible or aperiodic graphs. Again, lines 2-4 take place inside the objective function, however, alternatively, there is merit to these calculations taking place in the update rules of the evolutionary algorithms presented in the following subsections. A final remark concerns non-positivity of the entries of vector $\boldsymbol{\theta}$. As already mentioned, iterative normalization requires non-negativity to converge. If the optimizer of choice cannot enforce boundary restrictions on the optimization variables, then the user should enforce non-negativity by taking absolute or exponent values inside the objective functions.

3.6 Practical considerations

In our application, we transform each particle's (or candidate's) position with iterative normalization (Algorithm 1) as part of the objective function. When this is the case, one needs to examine the properties of the transformation and ensure that the velocity vectors of the swarm (or population) maintain their relevance with respect to the true objective of interest. For example, in the algorithm UPSO if the transformation replaced the entries of $\boldsymbol{\theta}_i$ with random numbers, the velocity vectors $\boldsymbol{v}_i(t+1)$ will not be able "simulate" the gradients of the true objective, and particles at subsequent iterations are unlikely to improve their positions. When one augments the particle equations with:

$$\boldsymbol{\theta}_i(t) \leftarrow T(\boldsymbol{\theta}_i(t)),$$

where $T(\cdot)$ is the transformation of choice, then the velocity vectors will maintain their relevance in the transformed space. We refer to this variant as the augmented EvoDSM. With a proper choice of $T(\cdot)$, one can refine search to their liking. We generally recommend the augmented version when it is applicable.

4 Experimental results

In this section, we aim at evaluating the effectiveness of our proposed framework. We aim at answering the following questions:

(a) Is our proposed methodology, EvoDSM, capable of optimizing for consensus?
(b) How does EvoDSM compare to the optimal symmetric matrix solution?
(c) Is EvoDSM able to find solutions in big problems in reasonable time? How does it scale?
(d) Can we optimize the topology of the graph with EvoDSM as well?

In order to answer the above questions, we perform extensive experiments with random graphs of increasing number of possible nodes. We generate 10 random undirected Erdos-Renui graphs for each $n \in \{20, 50, 80\}$ nodes respectively. All graphs were generated with probability equal to the connectivity threshold $\ln(n)/n$. We run each algorithm 20 times on each graph to assess average performance. The first 10 runs optimize for symmetric weights and the other 10 allow for asymetries. We used the default settings for CMA-ES which

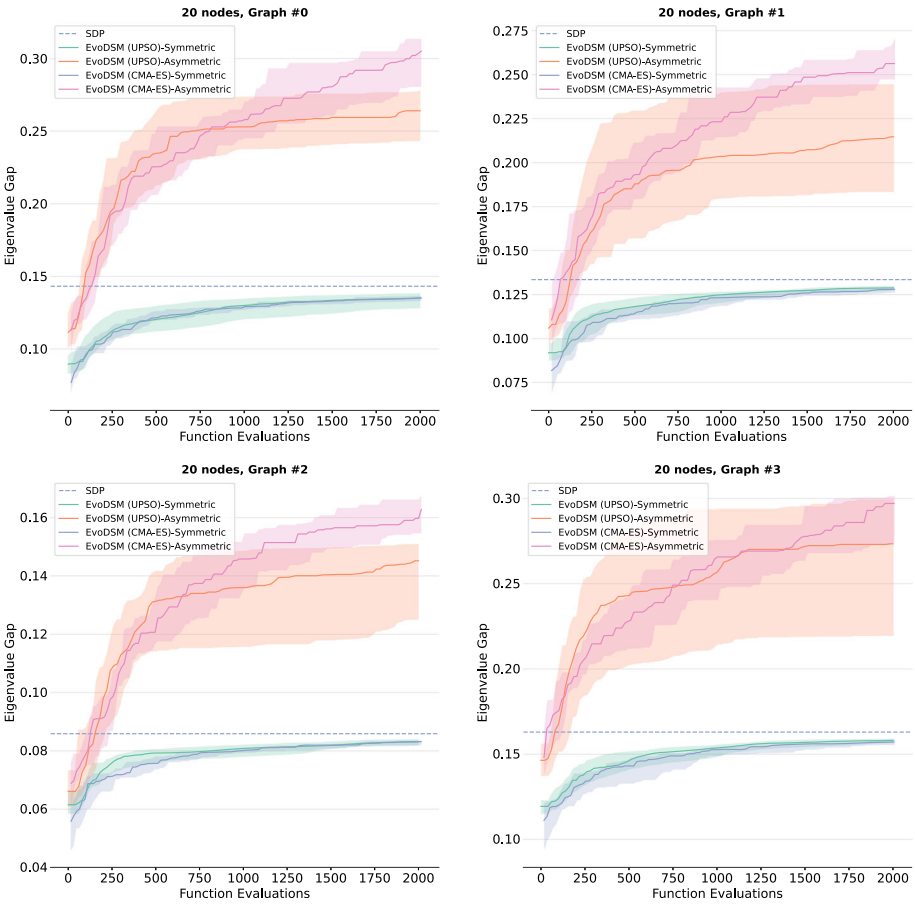


Fig. 1 Eigenvalue gap performance with graphs of 20 nodes. Solid lines are the median over 10 replicates and the shaded regions are the regions between the 5-th and 95-th percentiles

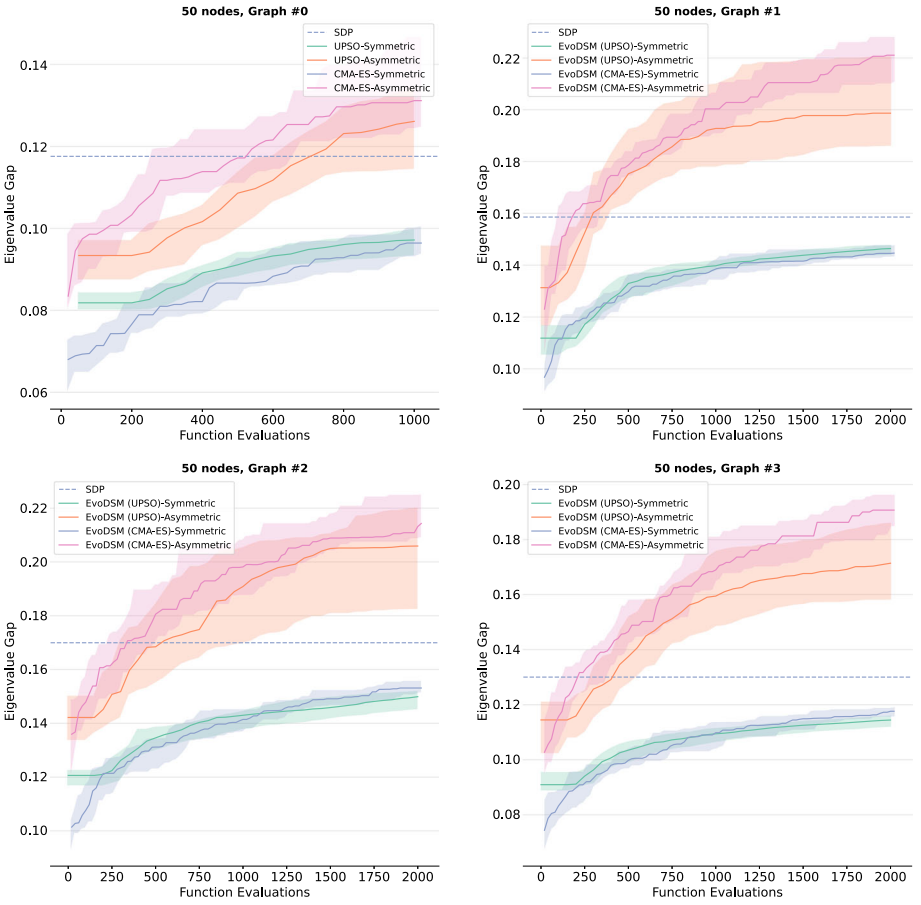


Fig. 2 Eigenvalue gap performance with graphs of 50 nodes. Solid lines are the median over 10 replicates and the shaded regions are the regions between the 5-th and 95-th percentiles

generates populations of approximately 20 individuals. For the particle optimizer we used n number of particles (as many as the number of nodes), way less than the usual convention (four times the dimension of the problem) requires.

4.1 Optimizing for consensus and comparison to SDP

In order to identify whether our proposed method is able to optimize for average consensus and how close it gets in finding the optimal value, we run 10 replicates of the optimization procedure for each graph with a budget of 2000 function evaluations. The results showcase that both EvoDSM (with UPSO and CMA-ES) are competitive at optimizing the weights of a symmetric matrix and get close to the optimal value as computed by [15, 24] (Fig. 1, 2, 3). The results also indicate that dropping the symmetry of the weight matrix allows us to find faster consensus convergence. We provide results for all 10 graphs in the appendix.

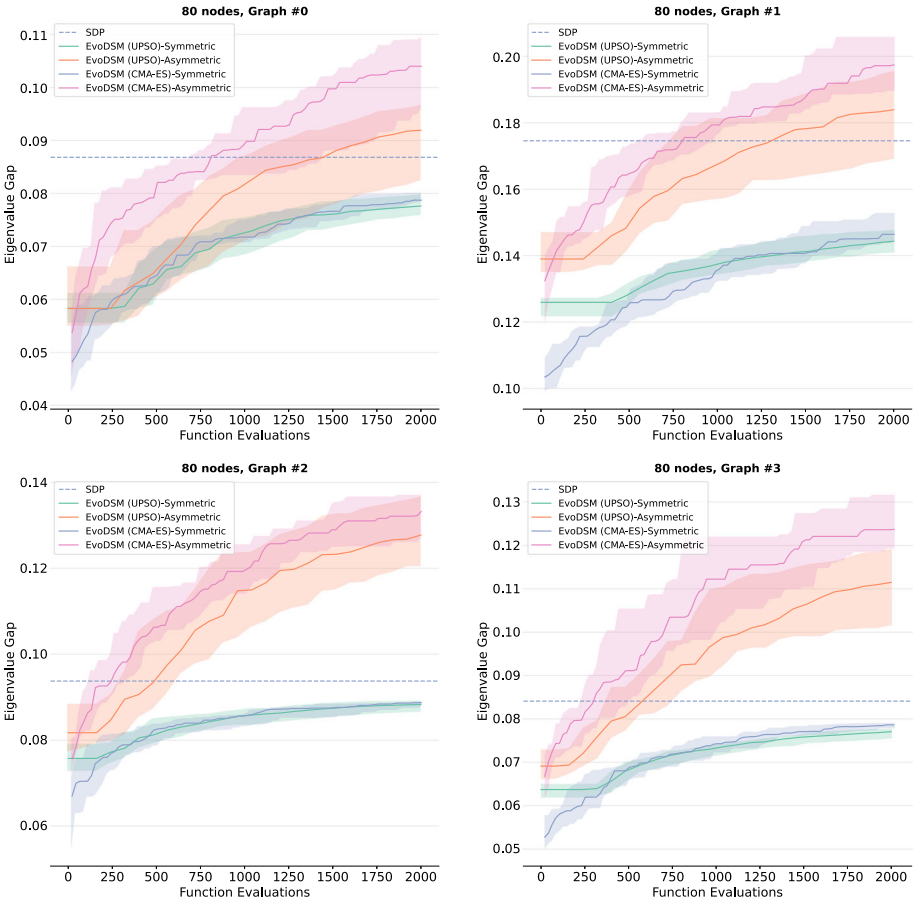


Fig. 3 Eigenvalue gap performance with graphs of 80 nodes. Solid lines are the median over 10 replicates and the shaded regions are the regions between the 5-th and 95-th percentiles

4.2 Scaling analysis

We are interested in inspecting how our proposed method scales with bigger problems/matrices. In order to do this, we make the following observations:

(a) UPSO (and any genetic algorithm) is a population-based algorithm with little to no overhead per iteration;
(b) CMA-ES, on the other hand, is a population-based algorithm that has considerable overhead per iteration that increases as the dimensions of the problem increase.

Having those observations in mind, we run both UPSO and CMA-ES without any parallelization and collect the total wall time for performing 2000 function evaluations. This procedure gives us the ability to compute timings for different parallelization schemes. Although the actual real wall times might be affected by many factors (e.g. cache misses, memory alignment, etc.), we were careful to provide estimates with the max idealized parallelization for each algorithm. In particular, UPSO uses a population of n (same as the number of nodes), and thus the maximum idealized gain from parallelization is n. On the contrary,

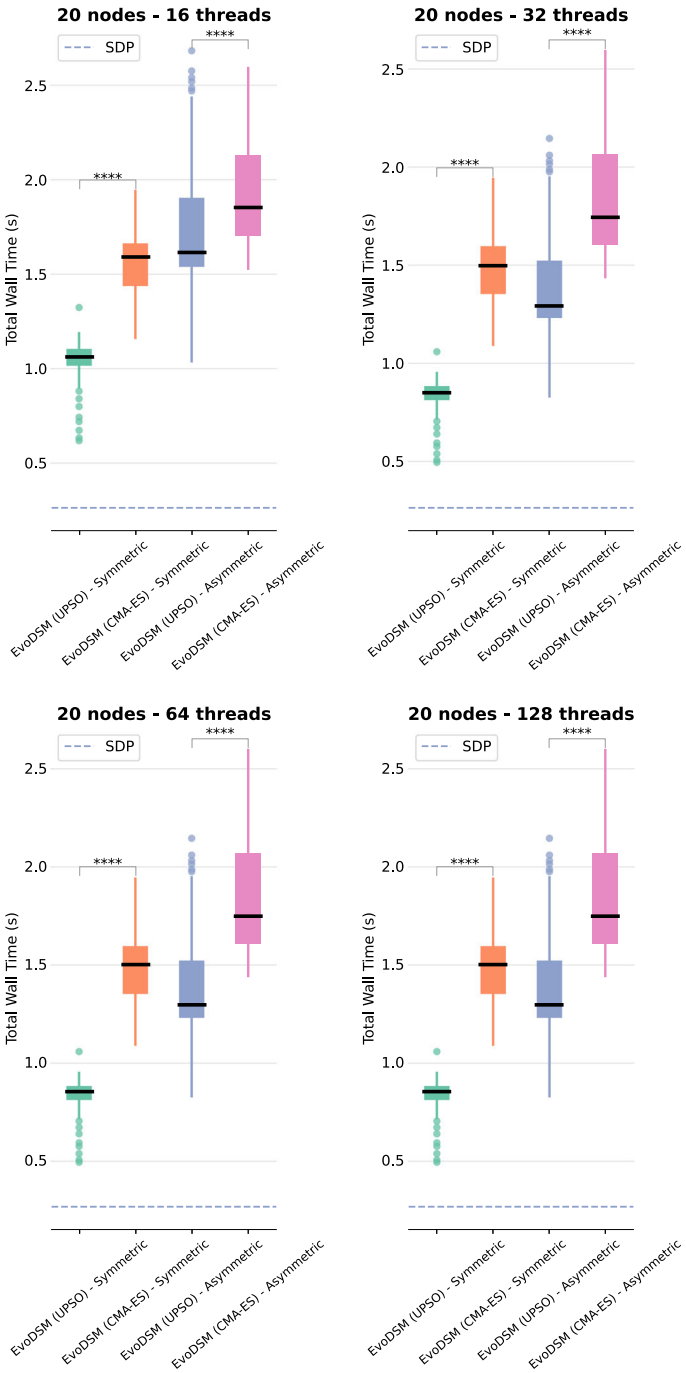


Fig. 4 Scaling of EvoDSM on graphs with 20 nodes (100 replicates). The box plots show the median (black line) and the interquartile range; the whiskers extend to the most extreme data points not considered outliers, and outliers are plotted individually. The number of stars indicates that the p-value of the Mann-Whitney U test is less than 0.05, 0.01, 0.001 and 0.0001 respectively

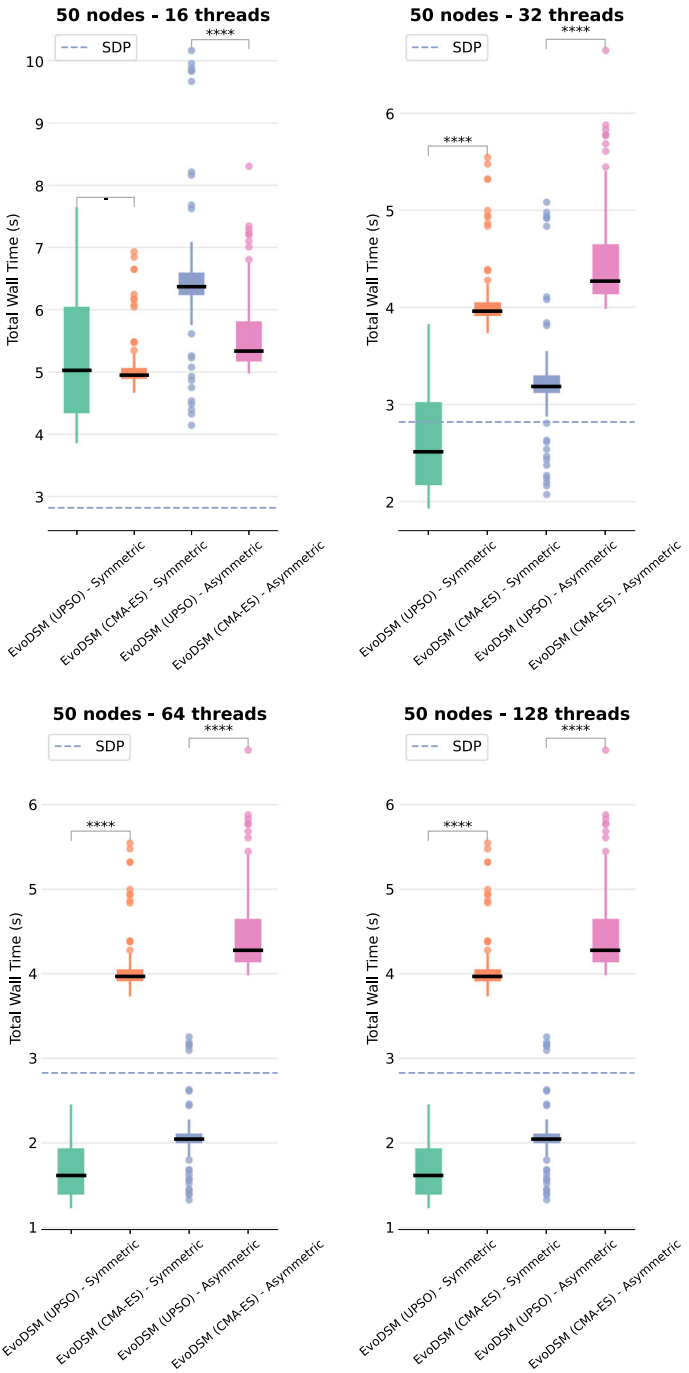


Fig. 5 Scaling of EvoDSM on graphs with 50 nodes (100 replicates). The box plots show the median (black line) and the interquartile range; the whiskers extend to the most extreme data points not considered outliers, and outliers are plotted individually. The number of stars indicates that the p-value of the Mann-Whitney U test is less than 0.05, 0.01, 0.001 and 0.0001 respectively

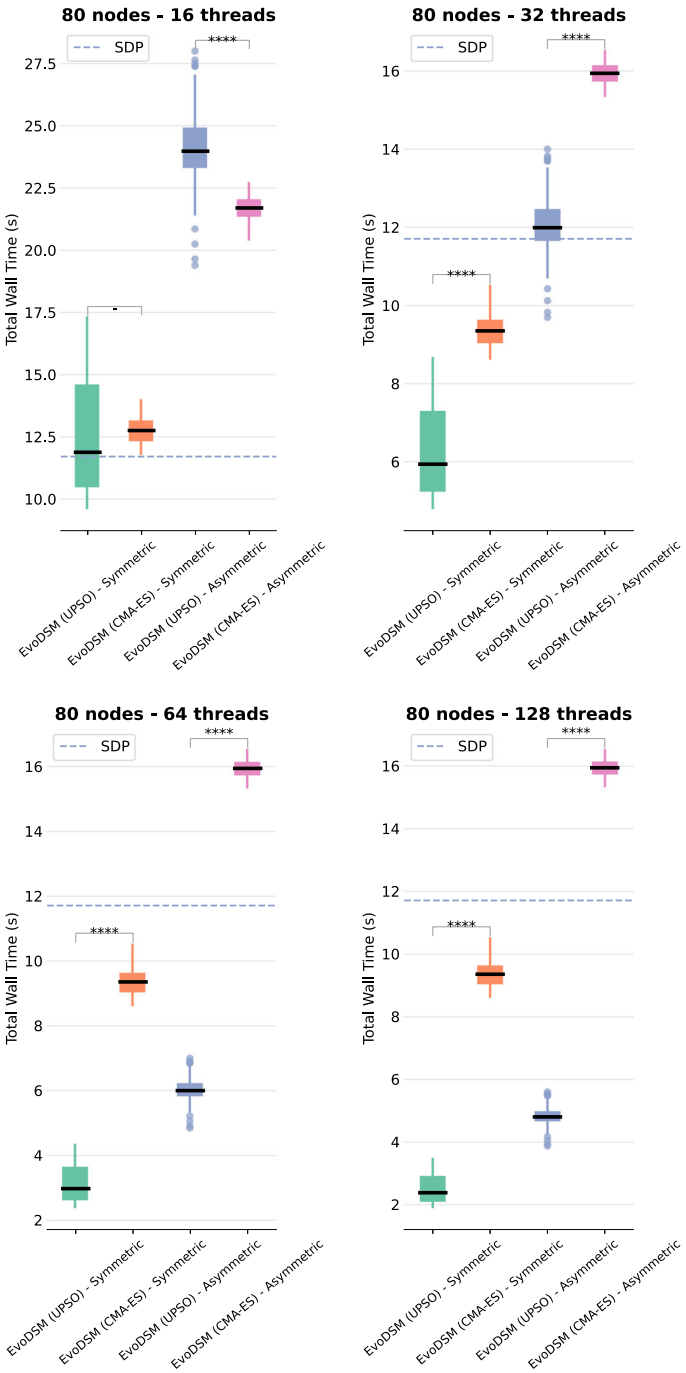
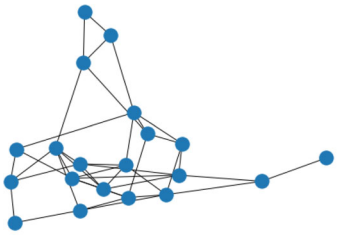
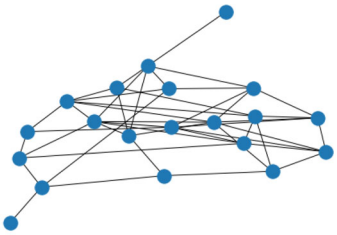


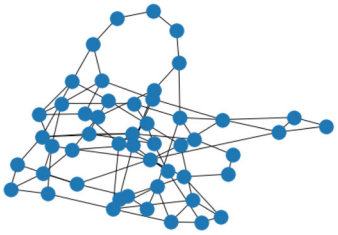
Fig. 6 Scaling of EvoDSM on graphs with 80 nodes (100 replicates). The box plots show the median (black line) and the interquartile range; the whiskers extend to the most extreme data points not considered outliers, and outliers are plotted individually. The number of stars indicates that the p-value of the Mann-Whitney U test is less than 0.05, 0.01, 0.001 and 0.0001 respectively



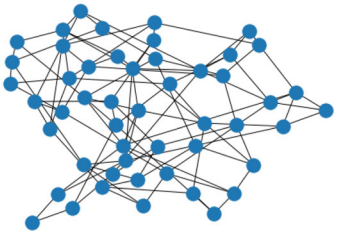
(a) 20 nodes, asymmetric, $\lambda_2 = 0.9$, 51 non-zero weights.



(b) 20 nodes, symmetric, $\lambda_2 = 0.9$, 93 non-zero weights.

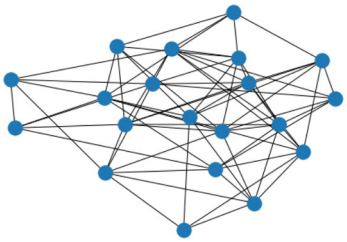


(c) 50 nodes, asymmetric, $\lambda_2 = 0.9$, 93 non-zero weights.

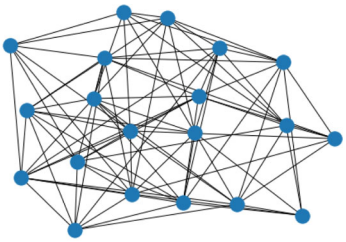


(d) 50 nodes, symmetric, $\lambda_2 = 0.9$, 202 non-zero weights.

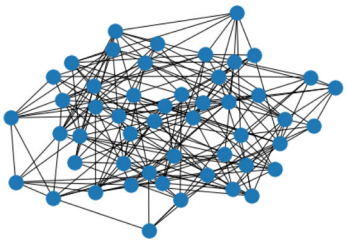
Fig. 7 Sparse graphs for $\lambda_t = 0.9$



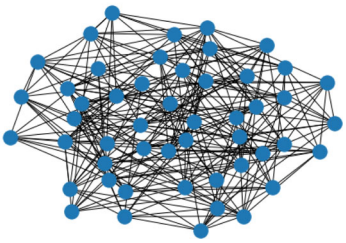
(a) 20 nodes, asymmetric, $\lambda_2 = 0.5$, 84 non-zero weights.



(b) 20 nodes, symmetric, $\lambda_2 = 0.5$, 179 non-zero weights.



(c) 50 nodes, asymmetric, $\lambda_2 = 0.5$, 231 non-zero weights.



(d) 50 nodes, symmetric, $\lambda_2 = 0.5$, 565 non-zero weights.

Fig. 8 Sparse graphs for $\lambda_t = 0.5$

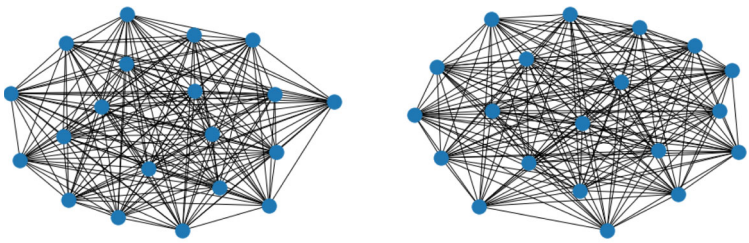
CMA-ES is using smaller population sizes, and thus the maximum idealized gain from parallelization is smaller. Overall, the main goal of this evaluation is to provide evidence that evolutionary algorithms can scale effectively with more CPU or GPU cores, and solve big problems in small amount of time.

Indeed, the results showcase exactly this, and furthermore we observe that algorithms like UPSO can scale more effectively since the overhead per iteration is minimal, it is helpful to use bigger populations and thus we can fully exploit the parallelization (Figs. 4, 5, and 6).

4.3 Optimizing for graph sparsification

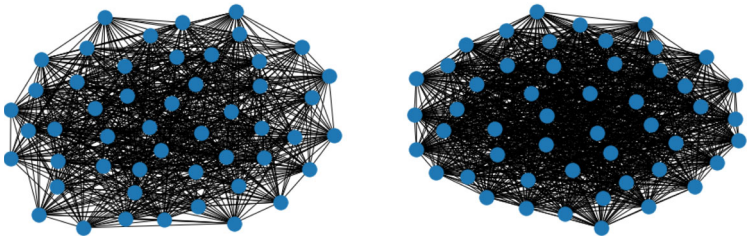
In this section we apply the objective function given by Algorithm 4. For these experiments, we only use UPSO. In particular, we use the augmented version of UPSO which performs lines 2-4 of Algorithm 4 to transform particle positions prior to objective evaluation. For asymmetric weight on n nodes, each particle maintains a vector of n^2 scalars in the interval $[0, 1]$. For symmetric weights, one can either maintain $\frac{n(n+1)}{2}$ variables corresponding to the upper triangular part of W, or further augment the UPSO update step with what corresponds to $W = (W + W^\top)/2$ to "symmetrize" the position. We performed experiments for $n \in \{20, 50\}$ and for eigenvalue-gap targets $\lambda_t \in \{0.1, 0.5, 0.9\}$. Each experiment was performed once for symmetric and once for asymmetric weights. In all cases, the number of particles used was n, the same as the number of nodes, and the algorithm was allowed to perform 4000 objective evaluations. Figures 7, 8, and 9 show the resulting graphs for each target respectively. The results are summarized in Table 1.

The results show that asymmetric weights outperform their symmetric counterpart in terms of sparsity in every experiment. All weight matrices have reached their target with



(a) 20 nodes, asymmetric, $\lambda_2 = 0.1$, 335 non-zero weights.

(b) 20 nodes, symmetric, $\lambda_2 = 0.1$, 388 non-zero weights.



(c) 50 nodes, asymmetric, $\lambda_2 = 0.169$, 980 non-zero weights.

(d) 50 nodes, symmetric, $\lambda_2 = 0.1$, 2186 non-zero weights.

Fig. 9 Sparse graphs for $\lambda_t = 0.1$

Table 1 Sparsification Results Summary

	nodes	λ_t	λ_2 (actual)	Non-zero entries
Sym.	20	0.1	0.100	388
Asym.	20	0.1	0.100	335
Sym.	20	0.5	0.500	179
Asym.	20	0.5	0.500	84
Sym.	20	0.9	0.900	93
Asym.	20	0.9	0.900	51
Sym.	50	0.1	0.100	2186
Asym.	50	0.1	0.168	980
Sym.	50	0.5	0.500	565
Asym.	50	0.5	0.500	231
Sym.	50	0.9	0.900	202
Asym.	50	0.9	0.900	93

the exception of Fig. 9(c). By inspection we can see that the graphs with the "slow" target ($\lambda_2 = 0.9$) yields single neighbor nodes and/or graphs with larger diameter (Fig. 7 (a),(b) respectively). For moderate speed ($\lambda_2 = 0.5$) we get intricately connected structures and for high speeds ($\lambda_2 = 0.1$) are dense.

5 Conclusion & future work

We developed the EvoDSM framework which enables evolutionary algorithms to optimize arbitrary, positive, doubly stochastic matrices. We consider the problem of minimizing the second largest eigenvalue magnitude (SLEM), which is known to be a convex problem for symmetric matrices but non-convex for asymmetric ones. Applications can be found in distributed systems and Markov Chain Monte Carlo methods. Our framework is enhanced by the use of an iterative normalization scheme that effectively maps the positions of the particles/candidates to the set of positive doubly stochastic matrices, and does not change the sparsity pattern.

We performed experiments using the *Unified Particle Swarm Optimizer* (UPSO), the *Covariance Matrix Adaptation Evolutionary Strategy* (CMA-ES) and a *Semi-Definite Problem* (SDP) solver for benchmarking. Results show that the optimized symmetric matrices approximate their optimal solutions and their asymmetric counterparts are able to reduce SLEM even further than the optimal symmetric case. We performed scaling analysis by computing total wall-times, showing that our framework is competitive with the SDP solver if implemented in a distributed setting.

Finally, to demonstrate the utility of EvoDSM in matrix discovery, we developed an objective function that promotes sparsity and attains a user-defined SLEM target. Our experiments indicate that, in all cases, asymmetric matrices achieve more sparsity than symmetric ones. Our framework is simple, effective, accessible, and allows for flexible objective formulations with no engineering overhead. Our results exhibit the viability and merit of attempting the non-convex SLEM minimization problem with our framework.

In future work, we aim at exploring different black-box optimizers as well as variations of the proposed objectives. Moreover, we will consider bigger graphs and attempt to identify

how the size of the graphs affects the performance of the proposed objective functions and parameterization. It would also be interesting to perform a comparative sensitivity analysis between UPSO and CMA-ES (and possibly other black-box optimizers) on the eigenvalue-gap maximization problem. Questions of interest would be (1) how do the sensitivity indices change between sparse and dense networks, (2) do sensitivity indices change as the size of the networks grows, (3) do sensitivity indices change on graphs generated by different ensembles and (4) can we find good parameter combinations in reasonable time.

Supplementary information We provide a supplementary file (pdf) containing the figures for all graphs generated as mentioned in Section 4.1. We also provide a supplementary file (a compressed archive) containing all the graphs generated for the experiments in Section 4.1 and Section 4.2.

Data Availibility The authors declare that all the data supporting the findings of this study are available within the article and its supplementary information files.

Declarations

Competing Interest The authors declare that there is no competing interest of any kind, directly or indirectly related to this work.

References

1. Boccaletti, S., Latora, V., Moreno, Y., Chavez, M., Hwang, D.-U.: Complex networks: Structure and dynamics. Phys. Rep. **424**(4–5), 175–308 (2006)
2. Cybenko, G.: Dynamic load balancing for distributed memory multiprocessors. J. Parallel Distrib. Comput. **7**(2), 279–301 (1989)
3. Belta, C., Kumar, V.: Abstraction and control for groups of robots. IEEE Transactions on robotics **20**(5), 865–875 (2004)
4. Yang, P., Freeman, R.A., Lynch, K.M.: Multi-agent coordination by decentralized estimation and control. IEEE Transactions on automatic control **53**(11), 2480–2496 (2008)
5. Peng, Z., Wang, J., Wang, D., Han, Q.-L.: An overview of recent advances in coordinated control of multiple autonomous surface vehicles. IEEE Transactions on industrial informatics **17**(2), 732–745 (2020)
6. Lin, C.-Y., Peng, W.-C., Tseng, Y.-C.: Efficient in-network moving object tracking in wireless sensor networks. IEEE Transactions on mobile computing **5**(8), 1044–1056 (2006)
7. Olfati-Saber, R.: Distributed kalman filter with embedded consensus filters. In: Proceedings of the 44th IEEE Conference on decision and control. IEEE, pp. 8179–8184 (2005)
8. Nedic, A., Ozdaglar, A.: Distributed subgradient methods for multi-agent optimization. IEEE Transactions on automatic control **54**(1), 48–61 (2009)
9. Kia, S.S., Cortés, J., Martínez, S.: Distributed convex optimization via continuous-time coordination algorithms with discrete-time communication. Automatica **55**, 254–264 (2015)
10. Chandrasekhar, A.G., Larreguy, H., Xandri, J.P.: Testing models of social learning on networks: Evidence from two experiments. Econometrica **88**(1), 1–32 (2020)
11. Hadjicostis, C.N., Domínguez-García, A.D., Charalambous, T., et al.: Distributed averaging and balancing in network systems: with applications to coordination and control. Found. Trends® Syst. Control 5(2–3), 99–292 (2018)
12. Hadjicostis, C.N., Domínguez-García, A.D., Charalambous, T.: Distributed Averaging and Balancing in Network Systems. Now Publishers, Norwell, MA (2018)
13. Kia, S.S., Van Scoy, B., Cortes, J., Freeman, R.A., Lynch, K.M., Martinez, S.: Tutorial on dynamic average consensus: The problem, its applications, and the algorithms. IEEE Control systems magazine **39**(3), 40–72 (2019)

14. Stamouli, C.J., Bechlioulis, C.P., Kyriakopoulos, K.J.: Robust dynamic average consensus with prescribed transient and steady state performance. Automatica **144**, 110503 (2022)
15. Boyd, S., Diaconis, P., Xiao, L.: Fastest mixing markov chain on a graph. SIAM Rev. **46**(4), 667–689 (2004)
16. Parsopoulos, K.E., Vrahatis, M.N.: UPSO: A unified particle swarm optimization scheme. Lect. Ser. Computer and Comput. Sci. **1**, 868–873 (2004)
17. Parsopoulos, K.E., Vrahatis, M.N.: Parameter selection and adaptation in unified particle swarm optimization. Math. Comput. Model. **46**, 198–213 (2007)
18. Parsopoulos, K.E., Vrahatis, M.N.: Particle Swarm Optimization and Intelligence: Advances and Applications. Inform. Sci. Publ. (IGI Global), Hershey, PA, USA (2010)
19. Cleghorn, C.W., Stapelberg, B.: Particle swarm optimization: Stability analysis using N-informers under arbitrary coefficient distributions. Swarm Evol. Comput. **71**, 101060 (2022)
20. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evol Comput. **9**, 159–195 (2001)
21. Lehman, J., Clune, J., Misevic, D., Adami, C., Altenberg, L., Beaulieu, J., Bentley, P.J., Bernard, S., Beslon, G., Bryson, D.M., et al.: The surprising creativity of digital evolution: A collection of anecdotes from the evolutionary computation and artificial life research communities. Artif. Life **26**(2), 274–306 (2020)
22. DeGroot, M.H.: Reaching a consensus. J. Am. Stat. Assoc. **69**(345), 118–121 (1974)
23. Levin, D.A., Peres, Y.: Markov Chains and Mixing Times, vol. 107. American Mathematical Society, Rhode Island, United States (2017)
24. Xiao, L., Boyd, S.: Fast linear iterations for distributed averaging. Syst. & Control Lett. **53**(1), 65–78 (2004)
25. Overton, M.L., Womersley, R.S.: On minimizing the special radius of a nonsymmetric matrix function: Optimality conditions and duality theory. SIAM J. Matrix Anal Appl. **9**(4), 473–498 (1988)
26. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Transactions on evolutionary computation **6**(2), 182–197 (2002)
27. Parsopoulos, K.E., Vrahatis, M.N.: Recent approaches to global optimization problems through particle swarm optimization. Nat. Comput. **1**, 235–306 (2002)
28. Hansen, N.: The CMA Evolution Strategy: A Comparing Review. Springer, New York City, United States (2006). https://doi.org/10.1007/3-540-32494-14
29. Beyer, H.-G., Schwefel, H.-P.: Evolution strategies-a comprehensive introduction. Nat. Comput. **1**(1), 3–52 (2002)
30. Hansen, N.: Benchmarking a BI-population CMA-ES on the BBOB-2009 function testbed. In: GECCO (2009)
31. Chatzilygeroudis, K., Vassiliades, V., Stulp, F., Calinon, S., Mouret, J.- B.: A survey on policy search algorithms for learning robot controllers in a handful of trials. IEEE Transactions on robotics 36, 328–347 (2020
32. Hansen, N.: Towards a new evolutionary computation. Stud. Fuzziness Soft Comput. **192**, 75–102 (2006)
33. Sinkhorn, R., Knopp, P.: Concerning nonnegative matrices and doubly stochastic matrices. Pac. J. Math. **21**(2), 343–348 (1967)