Particle swarm optimization and Hodge decomposition on graphs

Ichio Kikuchi¹, Akihito Kikuchi² *

¹*Internationales Forschungszentrum für Quantentechnik*

²*International Research Center for Quantum Engineering, Tokyo*

March 27, 2020

Abstract

This article illustrates the application of the particle swarm method to do numerical optimizations concerning the Hodge decomposition of the flows on graphs.

1 Introduction

The theory of Hodge Laplacians on Graph [1, 2] has two types of significance: the one is the relation to abstract mathematics, such as Hodge theory and cohomology; another is that which concerns the graphs – or the networks, which could incorporate the problems of the real world, provided that they are arranged on graphs. One can define the *flow* on the edges of graphs. In theory, the flow is decomposed into different types of sub-components: the flows propelled by the *gradient* and those which circumnavigate the closed circuits. In practice, to do such a decomposition is to solve an optimization in the linear space[1], to which we could apply a lot of algorithms. In this article, we choose the particle-swarm optimization (PSO) to process the numerical computation, since this heuristic algorithm has been proved to be the excellent solver for various optimization problems [3, 4, 5].

2 Particle Swarm Method

Let us review the standard PSO algorithm [4]; it mimics the behavior of a group of flying birds or other living creatures who seek some goals. The individuals participate in the search of the optima of the objective function, with the collaboration of the whole group: each individual seeks a better position by the random instinctive movement, although it lingers around the memory of the best position of the past. Besides, there is an ever-found best position for the group; its memory attracts the individuals; but the current best positions for individuals and the entire group shall vary places as the result of the continued

*akihito.kikuchi@gakushikai.jp (The corresponding author)

search. In the end, the whole group would swarm around the best position, which should be optimum. In the simulation, the individuals are impersonalized into the set of particles (with coordinates and velocities). The time-evolution of the particle system is executed in the discrete-time steps, similar to the conventional molecular dynamics simulation with the stochastic flavor; each particle will be attracted to the personal best point (concerning the objective function) and the group best point; the velocity of the particle is curbed by the random frictional parameter and every member of the group will cease to move and assembles at the unique best position.

The algorithm is summarized as follows. Let $\{\mathbf{s}_j(t)\}$ and $\{\mathbf{v}_j(t)\}$ be the coordinates and the velocities of particles ($j=1, \dots, n$) at the time t . Each particle keeps the memory of the past best position $\{\mathbf{p}_j\}$, at which the objective function has taken the best value. Likewise, the entire swarm keeps the past best position $\mathbf{g}$, at which some of the participants have achieved the best value of the objective function $f(\mathbf{x})$.

- 1 Initialization. Start from the set of particles, for which the velocity and the coordinate of each particle are chosen randomly.
- 2 Evaluation of objective function. For each particle, compute the objective function $f(\mathbf{s})$ at its coordinate $\mathbf{s}_j(t)$. The best position $\mathbf{s}_{best}$ in the present group (concerning the evaluated objective function) is found now. Then the past best positions of each particle $\{\mathbf{s}_j(t)\}$ and that of the entire group ($\mathbf{g}$) are updated as follows. For each particle, if $f(\mathbf{p}_j) < f(\mathbf{s}_j(t))$, then replace $\mathbf{p}_j$ with $\mathbf{s}_j(t)$; else keep $\mathbf{p}_j$. For the group, if $f(\mathbf{g}) < f(\mathbf{s}_{best})$, then replace $\mathbf{g}$ with $\mathbf{s}_{best}$; else keep $\mathbf{g}$.
- 3 Time evolution. The velocity and the coordinate are updated by the following formula.

$$\mathbf{v}_j(t+1) = W \cdot \mathbf{v}_j(t) + C_1 \cdot r_1 \cdot (\mathbf{p}_j - \mathbf{s}_j(t)) + C_2 \cdot r_1 \cdot (\mathbf{g} - \mathbf{s}_j(t)) \quad (1)$$

$$\mathbf{s}_j(t) = \mathbf{s}_j(t) + \chi \cdot \mathbf{v}_j(t) \quad (2)$$

The parameter w is the inertia which leads the group to a standstill in the end; C_1 , C_2 and χ are the parameter which determines the extent of the jump of the velocities and the positions from one step to the next; the alternation of the velocity and the positions are also regulated by random numbers r_1 and r_2 .

- 4 Go back to 2.

We have to judge the convergence of the optimization. In this work, when the velocities of the particles reduce below a certain threshold (and consequently, the group gathers around a certain point), we admit the convergence. Namely, for the vector velocities $(v_i^1, v_i^2, \dots, v_i^D)$ ($i = 1, \dots, n$) of n particles, we use the criterion

$$\frac{1}{nD} \sum_{i=1}^n \sum_j^D |v_i^j| < \epsilon.$$

In the computations of the example problems demonstrated later, the threshold of $\epsilon = 1.0^{-5} \sim 1.0^{-6}$ is always guaranteed.

3 Operators and Hodge Decomposition on Graphs

On a graph $G = (V, E)$ (with the nodes V and the edges E) we define a real-valued function $f : V \rightarrow \mathbb{R}$. As we use the index on the nodes ($\{i = 0, 1, 2, \dots, n\} \in V$), we denote the function by $f(i)$. In addition, we define the edge flow $X : V \times V \rightarrow \mathbb{R}$ and the triangular flow $\Phi : V \times V \times V \rightarrow \mathbb{R}$, which are defined on the edges $E := V \times V$ and the 3-cliques $T := V \times V \times V$ respectively. We require the latter two maps to be alternating, in the following sense:

$$X(i, j) = -X(j, i),$$

$$\Phi(i, j, k) = \Phi(j, k, i) = \Phi(k, i, j) = -\Phi(j, i, k) = \Phi(i, k, j) = \Phi(k, i, j).$$

With suitable weights w_i, w_{ij}, w_{ijk} , we define the inner product on V, E, T as follows

$$\langle f, g \rangle_V = \sum_{i=1}^n w_i f(i) g(i),$$

$$\langle X, Y \rangle_E = \sum_{i < j}^n w_{ij} X(i, j) Y(i, j),$$

$$\langle \Phi, \Psi \rangle_T = \sum_{i < j < k}^n w_{ijk} \Phi(i, j, k) \Psi(i, j, k).$$

Now we can define the operators on the graphs:

- $(\text{grad } f)(i, j) = f(j) - f(i)$.
- $(\text{curl } X)(i, j, k) = X(i, j) + X(j, k) + X(k, i)$. By convention, when i and j are not connected by an edge, then $(\text{grad } f)(i, j) = 0$. Likewise, when $\{ij\}$, $\{jk\}$ and $\{ki\}$ do not form a triangle in the graph, $(\text{curl } X) = 0$.

By means of the inner product, we define the dual δ^* to an operator δ as follows:

$$\langle \delta^* X, Y \rangle := \langle X, \delta Y \rangle.$$

From this formula, the duals to *grad* and *curl* are given by

- $(\text{div } X)(i) = \sum_{j=1}^n \frac{w_{ij}}{w_i} X(i, j)$
- $(\text{curl}^* \Phi)(i, j) = \sum_{k=1}^n \frac{w_{ijk}}{w_{ij}} \Phi(i, j, k)$

We define the combinatorial Laplacian Δ_k :

- $\Delta_0 = -\text{div} \circ \text{grad}$,
- $\Delta_1 = \text{curl}^* \circ \text{curl} - \text{grad} \circ \text{div}$.

Let C^1 be the linear space of the functions defined on the edges of a graph. It is decomposed into the direct sum of three subspaces [Hodge decomposition].

$$C^1 = \text{im}(\text{grad}) \oplus \ker(\Delta_1) \oplus \text{im}(\text{curl}^*)$$

The first, the second, and the third components of the decomposition are the subspaces of *gradient flows*, *harmonic flows*, and *curl flows*. The decomposition is done with the analogy of the vector calculus in the Euclidean space.

In addition, the direct sums of the pairs of the subspaces are the kernels of some operators:

$$\text{im}(\text{grad}) \oplus \ker(\Delta_1) = \ker(\text{curl})$$

$$\ker(\Delta_1) \oplus \text{im}(\text{curl}^*) = \text{im}(\text{div})$$

For each $\bar{Y} \in C^1$, the gradient and curl flows are determined respectively, as the solution to the following optimization problems.

* Gradient flows

$$\min_s \|\text{grad } s - \bar{Y}\|.$$

* Curl flows

$$\min_{\Phi} \|\text{curl}^* \Phi - \bar{Y}\|.$$

Hence the decomposition problems are equipped with the form to which numerical computations are applicable. The most plausible and accurate algorithm would be that of numerical linear algebra. But we employ a lazier and heuristic way; namely, we try to solve them using particle swarm optimization.

We warn the reader against this: owing to the reason that the decomposed components belong to the kernels of some operators, the optimization problem is replaced with the linear equation in the projected space. Thereby we could solve

$$\Delta_0 s = -\text{div} \bar{Y},$$

and

$$\text{curl}(\text{curl}^*(\Phi)) = -\text{curl} \bar{Y},$$

because the operators div and curl could get rid of incongruent parts. In this style, however, the nested operators take the more complicated forms, and in such circumstances, the computational cost for the repeated operations shall increase. Besides, the effort to minimize the residue allows us to investigate the problem from various standpoints; for example, we deal with the problem by sparse modeling; or we could manage to process the graphs on which part of the data is missing. For these reasons, we employ the PSO to solve directly the optimization problem (without projection) presented at first.

Remark. The arguments in the above are based upon *homology*. A k -dimensional cochain is the function f defined on $(k+1)$ -tuples (the set of vertices) on the complex K , which is *asymmetric* with respect to the permutation σ

$$f(i_{\sigma(0)}, i_{\sigma(1)}, \dots, i_{\sigma(k)}) = \text{sign}(\sigma) f(i_0, i_1, \dots, i_k).$$

The set of k -dimensional cochain is denoted by $C^k(K, \mathbb{R})$. The k -th *coboundary operator* is defined by

$$(\delta_k f)(i_0, i_1, \dots, i_{k+1}) := \sum_{j=0}^{k+1} (-1)^j f(i_0, \dots, i_{j-1}, i_{j+1}, \dots, i_{k+1}).$$

This definition leads to the property

$$\delta_k \circ \delta_{k-1} = 0.$$

We define the k th *cohomology group* of the graph G be the quotient vector space

$$H^k(G) = \ker(\delta_k) / \text{im}(\delta_{k-1}).$$

As for the operators defined on graphs, we have

$$\text{grad} = \delta_0,$$

$$\text{div} = -\delta_0^*,$$

$$\text{curl} = \delta_1,$$

$$\text{curl}^* = \delta_1^*.$$

The general statement of Hodge decomposition theorem is given by the formula:

$$C^k(K, \mathbb{R}) = \text{im}(\delta_{k-1}) \oplus \ker(\Delta_k) \oplus \text{im}(\delta_k^*)$$

in which the combinatorial Laplacian is defined by

$$\Delta_k = \delta_k^* \circ \delta_k + \delta_{k-1} \circ \delta_{k-1}^*.$$

In addition, the kernel of this operator is the intersection of two other kernels:

$$\ker(\Delta_k) = \ker(\delta_k) \cap \ker(\delta_{k-1}^*).$$

The special case of $k = 2$ is called Helmholtz decomposition theorem, to which we apply the PSO.

As for the rigorous background, see [2] and references therein.

4 Numerical Experiments

In this section, we report on the result of the numerical experiments. We show that the PSO can decompose flows on graphs into the aforementioned three distinct types with promptitude. The example problem is that given in Table 1, which concerns the currency exchange market among seven great economical powers.

To compute the cohomology we replace the numerical values in the tables with the logarithms; that is to say, the division (that of the ratio between two different currencies) is replaced with the subtraction:

$$X_{ij} = P_i / P_j \Rightarrow \log X_{ij} = \log P_i - \log P_j$$

By this transformation, we infer P_i and P_j from the logarithm of the data X_{ij} on the edge i, j , although the indeterminacy of P_i ever remains up to a certain constant. In addition, concerning the inner product and the operators on graphs, we set the weights (w_{ij}, w_{ijk}) on the existing edges and 3-cliques to be 1.

	USD	JPY	EUR	CAD	GBP	AUD	CHF
1 USD =	1.0000	114.6700	0.6869	0.9187	0.4790	1.0768	1.1439
1 JPY =	0.0087	1.0000	0.0060	0.0080	0.0042	0.0094	0.0100
1 EUR =	1.4558	166.9365	1.0000	1.3374	0.6974	1.5676	1.6653
1 CAD =	1.0885	124.8177	0.7477	1.0000	0.5214	1.1721	1.2451
1 GBP =	2.0875	239.3791	1.4340	1.9178	1.0000	2.2478	2.3879
1 AUD =	0.9287	106.4940	0.6379	0.8532	0.4449	1.0000	1.0623
1 CHF =	0.8742	100.2448	0.6005	0.8031	0.4188	0.9413	1.0000
Universal equivalent	1.7097	0.0149	2.4890	1.8610	3.5691	1.5878	1.4946

Table 1: The currency exchange rate table, taken from JIANG et al[1]. The corresponding graph is complete, namely, there is no missing link between any two of the nodes.

There are two remarkable features of this problem.

- As the exchange rate table correctly reflects the ratio of the values of the currencies, it is free from curl-flows.
- As the graph is complete, the harmonic component is absent. (This is less vivid, because the verification requires a numerical computation, although it is a mathematically rigorous result.)

We prepare several example problems from the table.

- (1) Do optimization for the gradient flow and curl flow of the table as it is.
- (2) Add the random noise to the data in the table. Then do the optimization.
- (3) Cut some of the edges so that the graph is not complete. Then do the optimization.
- (4) Do the optimization using various types of the norm. (For problems (1)-(3), shall we use l_2 norm. Here we examine other possibilities.)

Let us see the results.

As for the problem (1), Table 2 is computed by the optimization of the gradient flow $\text{grad}(s)$ and it shows the ratio of $\exp(-s(i))/\exp(-s(\text{USD}))$, where $i = \text{USD}, \text{JPY}, \text{EUR}, \text{CAD}, \text{GBP}, \text{AUD}, \text{CHF}$, after the different simulations with the change of the number of the sampling particles ($N = 5, 10, 50, 100, 250$). In the simulation, we set the parameters as follows $W = 0.7, C_1 = 1.4, C_2 = 1.4, \chi = 1.0$. The initial positions ($s(i)$) and the velocities ($v(i)$) of the particles are randomly distributed in the intervals in each coordinate $i = 1, \dots, 7$ such that $s_{init}(i) \in (-4, 4)$ and $v_{init}(i) \in (-2, 2)$. These parameters and the initial configurations are employed commonly in the other problems. To get rough estimation, it is enough to use 10~50 particles. In case of small number of sampling particles, the total group stops to advance before arriving at the optimum; they have lost the velocities on the way and had made a concentration at a point, which is not the optimum.

	USD	JPY	EUR	CAD	GBP	AUD	CHF	Residue
5ptcl	1.0000	0.0205	0.2002	0.5152	3.4591	0.8082	1.7959	83.0329
10ptcl	1.0000	0.0089	1.4741	1.0969	2.1857	0.9413	0.8729	0.0126
50ptcl	1.0000	0.0087	1.4555	1.0887	2.0860	0.9286	0.8739	0.0000
100ptcl	1.0000	0.0087	1.4555	1.0887	2.0860	0.9286	0.8739	0.0000
250ptcl	1.0000	0.0087	1.4555	1.0887	2.0860	0.9286	0.8739	0.0000
SOL	1.0000	0.0087	1.4558	1.0885	2.0876	0.9287	0.8715	

Table 2: The result of the optimization for the gradient flow. Each row of the table shows the ratio of $\exp(-s(i))/\exp(-s(\text{USD}))$, where $i = \text{USD}, \text{JPY}, \text{EUR}, \text{CAD}, \text{GBP}, \text{AUD}, \text{CHF}$. The rows, from the second to the third, show the different simulations computed with the change of the number of the sampling particle($N = 5, 10, 50, 100, 250$). The seventh row [SOL] shows the ratios of the universal equivalent, relative to USD, computed from Table 1.

As for the problem (2), Figure 1 shows the convergence of the optimization. The entire optimization employs the same initial setting as in the problem (1). For the gradient optimization, the residue remains non-zero; this is because of the presence of the noises. The optimization, however, can separate the gradient flow from the other components. To the remaining residue, the optimization for the curl* flow is applied. The residue, in this case, reduces to zero; namely, the residue of the gradient optimization is the curl flow. This is the numerical evidence of the completeness of the graph, by which no harmonic flow is permitted[2].

Figure 1: The reduction of residues in the optimizations. The minimization is at first done for $\|\text{grad}(s) - \bar{Y}\| (= R)$ and then for $\|\text{curl}^*(\Phi) - \bar{R}\|$. Both results are shown in one panel, denoted by two lines (OPTGRAD and OPTCURL).

As for problem (3), we assume a circular flow, in which the closest neighbors only do the trade. The model is given in Table 3; the random number as the logarithmic exchange is placed in the each edge and the matrix is skew-symmetric.

The typical residue of the gradient flow optimization is given in Table 4. As the graph

	USD	JPY	EUR	CAD	GBP	AUD	CHF
USD	–	Rnd ₁	–	–	–	–	–Rnd ₇
JPY	–Rnd ₁	–	Rnd ₂	–	–	–	–
EUR	–	–Rnd ₂	–	Rnd ₃	–	–	–
CAD	–	–	–Rnd ₃	–	Rnd ₄	–	–
GBP	–	–	–	–Rnd ₄	–	Rnd ₅	–
AUD	–	–	–	–	–Rnd ₅	–	Rnd ₆ –
CHF	Rnd ₇	–	–	–	–	–Rnd ₆	–

Table 3: The flow on the circle. We set random numbers Rnd₁,...Rnd₇, on the edges connecting adjacent nodes.

contains no 3-clique, the residue lies in the kernel of the curl operator. it has no contribution from curl. After removing the gradient flow, we obtain a matrix like this. As one can check easily (by adding the values of two connected edge), the residue lies in the kernel of the div operator.

	USD	JPY	EUR	CAD	GBP	AUD	CHF
USD	–	A	–	–	–	–	$-A$
JPY	$-A$	–	A	–	–	–	–
EUR	–	$-A$	–	A	–	–	–
CAD	–	–	$-A$	–	A	–	–
GBP	–	–	–	$-A$	–	A	–
AUD	–	–	–	–	$-A$	–	A
CHF	A	–	–	–	–	$-A$	–

Table 4: The typical residue to the gradient flow optimization applied to the circular graph of Table 3.

As for problem (4), we use the lasso-type norm with the parameter λ

$$\min_s \frac{1}{2\lambda} \|\text{grad}(s) - \bar{Y}\|_2^2 + \|\text{grad}(s)\|_1. \quad (3)$$

We do the optimization for the same case as the problem (1), without noise added to the data. We use 100 sampling particles and the same parameter set of other variables as that problem. The result is given in Table 5, in which the computed values are presented without exponential correction ($x \rightarrow \exp(-x)$). When λ is small, we obtain sparse solutions, in the sense that the entries in the solution vector tend to draw together at certain values. For example, $\lambda = 0.1$, the numerical values in the row takes only two alternatives. And the sparsity of the solution (in the aforesaid sense) is eminent up to $\lambda = 1.0$. When λ is large, however, the result is almost the same as the computation in problem (1), which is given in the bottom row. This result ascertains the applicability of the particle swarm optimization on graphs as the tool of the *sparse modeling* [6]. In all of the computed solutions, the component for JPY always dominates others. This domination is due to the quite small magnitude of JPY in the source of the currency exchange table (Table 1). This computation suggests that the optimization with the Lasso norm would delineate the rough profile of the flow.

λ	USD	JPY	EUR	CAD	GBP	AUD	CHF	Residue
0.1	0.0159	2.4216	0.0159	0.0159	0.0159	0.0159	0.0159	227.0137
0.2	-0.9711	2.6426	-0.9711	-0.9711	-0.9711	-0.9711	-0.9711	136.1280
0.5	-1.5072	2.8475	-1.5548	-1.5072	-1.7777	-1.5072	-1.5072	62.7701
1.0	-0.0723	4.4559	-0.3027	-0.0838	-0.5913	-0.0723	-0.0723	33.2156
10	-1.7891	2.9302	-2.1501	-1.8670	-2.5029	-1.7221	-1.6686	3.5015
100	-1.8266	2.9120	-2.2005	-1.9109	-2.5597	-1.7532	-1.6932	0.3520
1000	-1.8872	2.8533	-2.2624	-1.9722	-2.6222	-1.8132	-1.7526	0.0353
1000000	-1.8982	2.8425	-2.2736	-1.9832	-2.6335	-1.8241	-1.7635	0.0001

Table 5: The optimization for grad(s) for the same data as the problem (1), with the norm of Eq. (3). The result of $s(i)$ (i =USD, JPY, EUR, CAD, GBP, AUD, CHF) computed from the logarithms of the currency exchange table (Table 1) is presented here. The numerical values of the norm (namely, the residue) are also given at the rightmost column.

We can use other norms likewise.

* Standard Lasso

$$\min_s \frac{1}{2\lambda} \|\text{grad}(s) - \bar{Y}\|_2^2 + \|s\|_1. \quad (4)$$

* l_1 Residue

$$\min_s \|\text{grad}(s) - \bar{Y}\|_1. \quad (5)$$

The norm for the standard Lasso by Eq. (4) and that for l_1 norm by Eq.(5) are effective in computing sparse solutions aimed by the norm designs, as is shown in Table 6 and 7.

	USD	JPY	EUR	CAD	GBP	AUD	CHF	Residue
Standard Lasso $\lambda = 0.1$	0.0000	4.5847	-0.1830	-0.0000	-0.5444	-0.0000	0.0000	28.3404
l_1	0.2321	4.2905	-0.4941	-0.3924	-1.0996	-0.4955	-0.3914	15.1239
l_2	-0.0032	4.3510	-0.4344	-0.3272	-1.1767	-0.4049	-0.1236	9.6483

Table 6: A comparison of the gradient flow optimizations by different types of norms. The computations were done for the edge flow given by the logarithms of the table of the currency exchange market (Table 1), to which uniform random noises $\in (-1, 1)$ are added. Thereby the optimized $s(i)$ on the nodes are shown.

Residues by l_1

0.0000	0.0032	0.5618	-0.0264	-0.0012	-0.0217	1.3628
	0.0000	-0.0018	-0.6818	0.3619	0.0111	0.1219
		0.0000	-0.0686	0.5156	0.4242	-0.4972
			0.0000	-0.7765	-0.0095	-0.2165
				0.0000	0.9227	-0.0518
					0.0000	0.9237
						0.0000

Residues by l_2

0.0000	-0.2925	0.2669	-0.3270	-0.1594	-0.3476	0.8597
	0.0000	-0.0010	-0.6866	0.4995	-0.0191	-0.0854
		0.0000	-0.0743	0.6523	0.3932	-0.7053
			0.0000	-0.6341	-0.0349	-0.4190
				0.0000	0.7549	-0.3966
					0.0000	0.7465
						0.0000

Table 7: The difference between the optimizations by l_1 and l_2 norms, appearing in the residues $(\text{grad}(s) - \bar{Y})$ on the edges (i, j) . These residues are derived from the results of the computations of Table 6. The numerical values are placed in 7 by 7 matrices T_{ij} , according to the index: $i, j = (1:\text{USD}), (2:\text{JPY}), (3:\text{EUR}), (4:\text{CAD}), (5:\text{GBP}), (6:\text{AUD}), (7:\text{CHF})$. Due to the nature of the matrices, only the diagonals and the upper-left parts are given.

5 Summary and conclusion

In this article, we apply the particle swarm optimization to the Hodge decomposition of flows on graphs, which is formulated as the optimization. The numerical experiments show the efficacy of particle swarm optimization as the solver of the optimization (the Hodge decomposition) on graphs, as it can adroitly distinguish and separate the three different types of flows on a graph. The merit of the particle swarm optimization is the broad applicability to various problems. One of the reasons for this is the fact that it never requires the derivative of the objective function. In the case of Hodge decomposition on graphs, this merit permits us to do the sparse modeling using the l_1 or Lasso norm, as is explained in the example problem in this article. Although the particle swarm optimization is heuristic and it could not attain the highest accuracy in the comparison with other algorithms, it will serve us as the apt tool of analysis applicable to gross scale problems in the real world.

Acknowledgment

The sample programs used in the numerical experiments are now available in public from GitHub (www.github.com/kikuchiichio/OptGraph). Though we (the authors) disclaim the copyright and leave the programs in the public domain, free to distributions and modifications, we shall be grateful to you if you refer to this article in case you make scientific publications using them.

References

- [1] Xiaoye Jiang, Lek-Heng Lim, Yuan Yao, and Yinyu Ye. Statistical ranking and combinatorial hodge theory. *Mathematical Programming*, 127(1):203–244, 2011.
- [2] Lek-Heng Lim. Hodge laplacians on graphs. *arXiv preprint arXiv:1507.05379*, 2015.
- [3] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE, 1995.
- [4] Federico Marini and Beata Walczak. Particle swarm optimization (pso). a tutorial. *Chemometrics and Intelligent Laboratory Systems*, 149:153–165, 2015.
- [5] Saptarshi Sengupta, Sanchita Basak, and Richard Alan Peters. Particle swarm optimization: A survey of historical and recent developments with hybridization perspectives. *Machine Learning and Knowledge Extraction*, 1(1):157–191, 2019.
- [6] Trevor Hastie, Robert Tibshirani, and Martin Wainwright. *Statistical learning with sparsity: the lasso and generalizations*. CRC press, 2015.