Concurrency and Computation: Practice and Experience

WILEY

SPECIAL ISSUE PAPER OPEN ACCESS

Algebraic Connectivity Maximizing Regular Graphs: Special Case Analysis and Depth-First Search

Masashi Kurahashi[1] | Najd Salaani[2] | Tsuyoshi Migita[3] | Norikazu Takahashi[3]

[1]Graduate School of Environmental, Life, Natural Science and Technology, Okayama University, Okayama, Japan | [2]Polytech Sorbonne, Sorbonne University, Paris, France | [3]Faculty of Environmental, Life, Natural Science and Technology, Okayama University, Okayama, Japan

Correspondence: Norikazu Takahashi (takahashi@okayama-u.ac.jp)

Received: 18 May 2025 | **Revised:** 22 September 2025 | **Accepted:** 9 October 2025

Funding: This work was supported by JSPS KAKENHI, grant number JP25K03196.

Keywords: algebraic connectivity | depth-first search | optimization | pruning | regular graph

ABSTRACT

The algebraic connectivity is an indicator of how well connected a graph is. It also characterizes the convergence speed of some dynamic processes over networks. In this paper, taking into account that homogeneous networks are modeled as regular graphs, we tackle the following problem: given a pair (n, k) of positive integers such that k is less than n and kn is an even number, find a k-regular graph with n vertices that have the maximum algebraic connectivity. We first consider some special cases and derive solutions through theoretical analysis. We next present depth-first search algorithms for solving the problem, which reduce the search space by making use of some known properties of the regular graph and the algebraic connectivity. We also show the results of execution of the proposed algorithms for the values of n up to 12.

1 | Introduction

The algebraic connectivity (AC) [1] of a simple undirected graph, which is defined as the second smallest eigenvalue of the Laplacian matrix, is an indicator of how well connected the graph is. It is well known that the AC of any connected graph is positive, whereas that of any disconnected graph is zero. It is also well known that the AC of any incomplete graph does not exceed its vertex connectivity and edge connectivity [2]. Other properties of the AC have been extensively studied in various areas of discrete mathematics and combinatorial optimization [3].

In many applications, it is desirable to design a network which is robust against failures on nodes and links, and the AC is useful for measuring the robustness of the network. For example, the AC has been used to design air transportation networks [4, 5], the digital logistic networks [6], multilayer networks [7], weighted networks related to cooperative vehicle localization [8], and so on. The AC also characterizes the convergence speed of some dynamic processes over networks [9–13]. Thus many attempts have been made so far to find graphs with high AC values under some conditions such as order, size, degrees and diameter [14–18]. For example, Ogiwara et al. [16] showed that some well-known classes of graphs such as star graphs, cycle graphs, complete bipartite graphs and circulant graphs are AC maximizers or AC local maximizers under certain conditions.

Among various types of graphs, regular graphs are particularly important from the viewpoint of applications because homogeneous networks such as supercomputers and data center networks are often modeled as regular graphs [19]. Thus many authors have studied so far the AC of regular graphs [20–25]. For example, Olfati-Saber [22] gave an explicit formula for the AC of regular lattices, and proved the growth rate of the AC of

Ramanujan graphs. Very recently, Exoo et al. [24] derived attainable upper bounds on the AC of a regular graph in terms of its diameter and girth. Kolokolnikov [25] explicitly computed the AC of a large random semi-regular bipartite graph and compared it with the AC of a random regular graph with the same number of vertices and edges. However, despite having been studied for many decades, the properties of the AC of regular graphs are not fully understood.

In this paper, we focus our attention on regular graphs, and consider the following problem: given a pair (n, k) of positive integers such that k is less than n and kn is an even number, find a simple undirected k-regular graph with n vertices that has the maximum AC. Note that no other conditions than order and degree are assumed in this problem. This is an important difference from the problems considered in the literature [20–24]. We first show that if $k \in \{2, n-3, n-2, n-1\}$ or $k = n(p-1)/p$ with p being a divisor of n then the above problem can be solved analytically using some known results in the literature. We next present Depth-First Search (DFS) algorithms for solving the problem, which are exhaustive search but prune unnecessary search branches based on some properties of regular graphs and the AC, and explain how to implement the proposed algorithms. We then show the results of execution of the proposed algorithms for n up to 12, and characterize the regular graphs found using the proposed algorithms in terms of multipartite graphs.

Among various search algorithms for regular graphs [26, 27], we employ the DFS algorithms for the following reasons. First, they are simple and easy to implement using a stack. Second, the rooted binary tree used in the DFS algorithms has a good property that, when traversing the tree from the root to a leaf, the number of edges in the graph increases monotonically, which implies that the AC of the graph increases monotonically. Making use of this property, we can develop pruning techniques that greatly reduce the search space. Third, the DFS algorithms can be easily parallelized by assigning sub-trees to multiple CPU cores. It is shown through experiments that the parallel implementation can greatly reduce the execution time of the DFS algorithms.

Contributions of this paper and their relationship to existing work can be summarized as follows. First, exact solutions to the problem are obtained in some special cases through theoretical analysis, which contain some of the results given by Ogiwara et al. [16], Ishii and Takahashi [17], and Shahbaz et al. [23] as special cases. Second, new DFS algorithms that incorporate pruning techniques tailored to the problem are proposed, and the effectiveness of the proposed algorithms and their parallel implementations is experimentally demonstrated. Third, exact solutions to the problem are found for n up to 12 using the proposed DFS algorithms, whereas recent related studies have provided upper bound analysis [24] and probabilistic analysis [25].

A preliminary version of the present paper appeared in Proceedings of the 2024 Twelfth International Symposium on Computing and Networking [28]. However, the present paper includes many new results. First, a theoretical analysis for the case where $k = n-3$ is provided. Second, a pruning technique based on the graph isomorphism is introduced in the DFS algorithms and a new parallel implementation of the proposed DFS algorithms is developed. Third, new solutions for the case where $n = 12$ are obtained using the proposed algorithms.

2 | Problem Formulation

2.1 | Notations and Definitions

Throughout this paper, by a graph we always mean a simple undirected graph with a finite number of vertices. Let $G = (V, E)$ be a graph with the vertex set $V = \{1, 2, \ldots, n\}$ and the edge set $E = \{e_1, e_2, \ldots, e_m\}$. Each edge is expressed as an unordered pair of distinct vertices like $e_h = \{i, j\}\,(i \neq j)$. The number of vertices adjacent to (or the number of edges incident to) the vertex i is called the degree of i and denoted by k_i. If all vertices have the same degree k then G is called a k-regular graph. If G is a connected 2-regular graph then G is called a cycle graph and denoted by C_n. If every pair of distinct vertices is adjacent to each other then G is called a complete graph and denoted by K_n. If V can be partitioned into p disjoint subsets $V_1, V_2, \ldots, V_p$ and there is no edge connecting two vertices in the same subset then G is called a p-partite graph. In particular, G is called a bipartite graph when $p = 2$ and a tripartite graph when $p = 3$. If G is a p-partite graph and every vertex is adjacent to all vertices in different subsets then G is called a complete p-partite graph and denoted by $K_{n_1, n_2, \ldots, n_p}$ where $n_i = \mid V_i \mid$ for $i = 1, 2, \ldots, p$.

For a graph $G = (V, E)$ with $V = \{1, 2, \ldots, n\}$, the graph with the vertex set V and the edge set $\overline{E}$ such that $e_h = \{i, j\} \in \overline{E}\ (i \neq j)$ if and only if $e_h \notin E$ is called the complement graph of G and denoted by $\overline{G}$.

The Laplacian matrix of a graph $G = (V, E)$ is defined by $\boldsymbol{L} \coloneqq \boldsymbol{D} - \boldsymbol{A}$, where $\boldsymbol{A}$ is the adjacency matrix, of which the (i, j)-th and (j, i)-th entries take 1 if $\{i, j\} \in E$ and 0 otherwise, and $\boldsymbol{D} \coloneqq \mathrm{diag}(k_1, k_2, \ldots, k_n)$ is the degree matrix. Let $\lambda_1(G), \lambda_2(G), \ldots, \lambda_n(G)$ be the eigenvalues of $\boldsymbol{L}$ of G listed in increasing order. Since $\boldsymbol{L}$ is a positive semi-definite matrix, the smallest eigenvalue $\lambda_1(G)$ is nonnegative. Moreover, since $\boldsymbol{L}\mathbf{1} = \mathbf{0} = 0 \cdot \mathbf{1}$ where $\mathbf{1}$ is the vector of all ones, the smallest eigenvalue $\lambda_1(G)$ is always 0. The second smallest eigenvalue $\lambda_2(G)$ is called the algebraic connectivity (AC) [1] of the graph G. As its name suggests, the AC indicates how well connected G is. In particular, the AC is positive if and only if G is connected [1]. Also, the AC of any incomplete graph does not exceed its vertex connectivity and edge connectivity [2].

2.2 | Problem Statement

Let (n, k) be a pair of positive integers such that $2 \leq k \leq n-1$ and kn is an even number, and let $\mathcal{R}_{n,k}$ be the set of all k-regular graphs with the vertex set $V = \{1, 2, \ldots, n\}$. If the AC of a graph $G \in \mathcal{R}_{n,k}$ is greater than or equal to that of any other graph in $\mathcal{R}_{n,k}$, then we call G an Algebraic Connectivity Maximizing (ACM) regular graph in $\mathcal{R}_{n,k}$.

The problem we consider in this paper is formally stated as follows.

Problem 1. Given a pair (n, k) of positive integers such that $2 \le k \le n-1$ and kn is an even number, find an ACM regular graph in $\mathcal{R}_{n,k}$.

The AC value varies greatly even though the order n and the degree k are fixed. To see this, we consider the three regular graphs in $\mathcal{R}_{10,4}$ shown in Figure 1. The AC values of the graphs in Figure 1a–c are 1.859877, 2.533268 and 3, respectively. The graph in Figure 1b is obtained from the one in Figure 1a by applying a 2-switch [29]. To be more precise, removing two edges $\{1,4\}$ and $\{5,9\}$ from the graph in Figure 1a and then adding two edges $\{1,5\}$ and $\{4,9\}$, we obtain the graph in Figure 1b. Since a single 2-switch can increase the AC value from 1.859877 to 2.533268, one may expect that it can be further increased if we apply a 2-switch to the graph in Figure 1b appropriately. However, with the help of a computer program, we can verify that any 2-switch applicable to the graph in Figure 1b cannot increase the AC value. This means that a solution to Problem 1 cannot be found by local search based on 2-switch.

3 | Special Case Analysis

Problem 1 can be solved analytically in some special cases. We present in this section solutions for those cases.

First we consider the case where $k = 2$. In this case, the following result is obtained.

Proposition 1. *Let n be any integer greater than or equal to three. A graph G is an ACM regular graph in $\mathcal{R}_{n,2}$ if and only if G is the cycle graph C_n. The AC of the cycle graph is given by*

$$\lambda_2(C_n) = 2\left(1 - \cos\frac{2\pi}{n}\right). \tag{1}$$

Proof. Let G be a graph in $\mathcal{R}_{n,2}$. Then G consists of either a disjoint union of multiple cycles or a single cycle. In the former case, $\lambda_2(G) = 0$ because G is not connected. In the latter case, $\lambda_2(G)$ is given by the right-hand side of (1), as shown in Li et al. [12] for example, which is positive. Therefore G is an ACM regular graph in $\mathcal{R}_{n,2}$ if and only if G is the cycle graph. □

We next consider the case where $k = n-1$. In this case, the following result is immediately obtained because $\mathcal{R}_{n,n-1}$ consists only of the complete graph K_n.

Proposition 2. *Let n be any integer greater than or equal to two. The complete graph K_n is the unique ACM regular graph in $\mathcal{R}_{n,n-1}$. The AC of K_n is n.*

We then consider the case where $k = n-2$. In this case, the following result is obtained.

Proposition 3. *Let n be any even number greater than or equal to four. Any graph in $\mathcal{R}_{n,n-2}$ is an ACM regular graph in $\mathcal{R}_{n,n-2}$. The AC of such a graph is $n-2$.*

Proof. The complement graph $\overline{G}$ of any $G \in \mathcal{R}_{n,n-2}$ is a 1-regular graph, which is a disjoint union of $n/2$ complete graphs with two vertices. Hence the eigenvalues of the Laplacian matrix of $\overline{G}$ are $\lambda_i(\overline{G}) = 0$ for $i = 1, 2, \ldots, n/2$ and $\lambda_i(\overline{G}) = 2$ for $i = n/2+1, n/2+2, \ldots, n$. Taking this fact and the well-known relationship

$$\lambda_2(G) = n - \lambda_n(\overline{G}) \tag{2}$$

(see Fiedler [2] for example), we can state that all graphs in $\mathcal{R}_{n,n-2}$ have the same AC, which is $n-2$. □

We then consider the case where n is a proper multiple of some integer p greater than or equal to two and $k = n(p-1)/p$. In this case, the following result is obtained.

Proposition 4. *Let n be a proper multiple of some integer p greater than or equal to two. The complete p-partite graph $K_{n/p,n/p,\ldots,n/p}$ is an ACM regular graph in $\mathcal{R}_{n,n(p-1)/p}$ and the AC of $K_{n/p,n/p,\ldots,n/p}$ is $n(p-1)/p$.*

Proof. The statement is a special case of a theorem given by Ishii and Takahashi [17]. They proved that the complete p-partite graph $K_{n/p,n/p,\ldots,n/p}$ has the largest AC, which is $n - n/p = n(p-1)/p$, among all graphs consisting of n vertices and $m = n^2(p-1)/2p$ edges. Nevertheless, we provide a proof to make this paper self-contained.

It is well known that if G is not a complete graph then $\lambda_2(G)$ is upper bounded by the minimum degree [1]. Since the minimum degree of any graph in $\mathcal{R}_{n,n(p-1)/p}$ is $n(p-1)/p$, it suffices for us to show that $\lambda_2\left(K_{n/p,n/p,\ldots,n/p}\right) = n(p-1)/p$. The complement graph of $K_{n/p,n/p,\ldots,n/p}$ has p connected components which are isomorphic to $K_{n/p}$. Thus the eigenvalues of the Laplacian

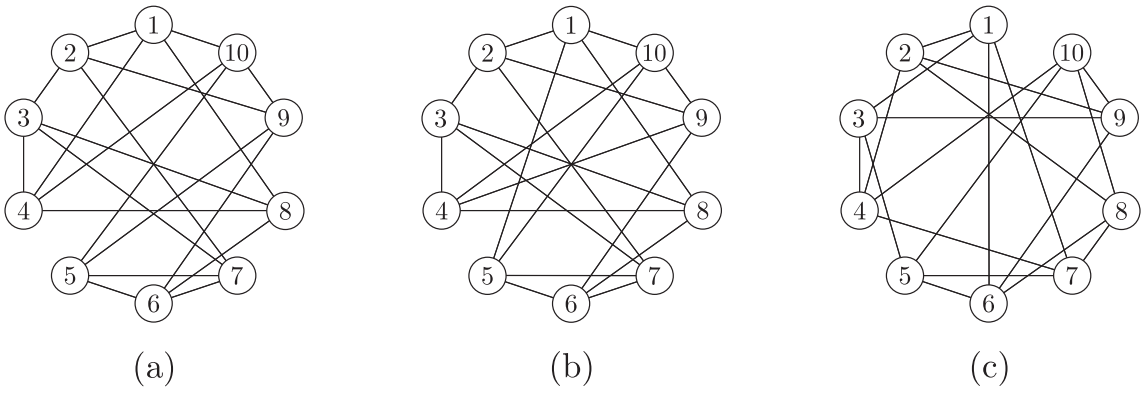

FIGURE 1 | Three 4-regular graphs with 10 vertices. The AC values of the graphs in (a), (b) and (c) are 1.859877, 2.533268 and 3, respectively.

matrix of the complement graph $\overline{K}_{n/p,n/p,\ldots,n/p}$ are given by

$$\lambda_i\left(\overline{K}_{n/p,n/p,\ldots,n/p}\right) = \begin{cases} 0, & i = 1, 2, \ldots, p, \\ n/p, & i = p+1, p+2, \ldots, n. \end{cases}$$

Let $\{\boldsymbol{v}_1 (= \mathbf{1}), \boldsymbol{v}_2, \ldots, \boldsymbol{v}_p\}$ and $\{\boldsymbol{v}_{p+1}, \boldsymbol{v}_{p+2}, \ldots, \boldsymbol{v}_n\}$ be orthogonal bases of the eigenspaces of the Laplacian matrix of $\overline{K}_{n/p,n/p,\ldots,n/p}$ associated with eigenvalues 0 and n/p, respectively. Then we have

$$\begin{aligned} \boldsymbol{L}\left(K_{n/p,n/p,\ldots,n/p}\right)\boldsymbol{v}_i &= \left(\boldsymbol{L}\left(K_n\right) - \boldsymbol{L}\left(\overline{K}_{n/p,n/p,\ldots,n/p}\right)\right) \\ \boldsymbol{v}_i &= \left(n\boldsymbol{I} - 11^\top\right)\boldsymbol{v}_i - \boldsymbol{L}\left(\overline{K}_{n/p,n/p,\ldots,n/p}\right) \\ \boldsymbol{v}_i &= \begin{cases} 0, & i = 1, \\ n\boldsymbol{v}_i, & i = 2, 3, \ldots, p, \\ (n - n/p)\boldsymbol{v}_i, & i = p+1, p+2, \ldots, n \end{cases} \end{aligned}$$

where $\boldsymbol{L}(G)$ denotes the Laplacian matrix of G and $\boldsymbol{I}$ is the identity matrix. Therefore, the eigenvalues of the Laplacian matrix of $K_{n/p,n/p,\ldots,n/p}$ are given by

$$\lambda_i\left(K_{n/p,n/p,\ldots,n/p}\right) = \begin{cases} 0, & i = 1, \\ n(p-1)/p, & i = 2, 3, \ldots, n-p+1, \\ n, & i = n-p+2, n-p+3, \ldots, n \end{cases}$$

which completes the proof. □

It is seen from Proposition 4 that if n is a positive even number then the complete bipartite graph $K_{n/2,n/2}$ is an ACM regular graph in $\mathcal{R}_{n,n/2}$ and $\lambda_2\left(K_{n/2,n/2}\right) = n/2$. For example, $K_{3,3}$ shown in Figure 2a is an ACM regular graph in $\mathcal{R}_{6,3}$. It is also seen that if n is positive and a multiple of three then the complete tripartite graph $K_{n/3,n/3,n/3}$ is an ACM regular graph in $\mathcal{R}_{n,2n/3}$ and $\lambda_2\left(K_{n/3,n/3,n/3}\right) = 2n/3$. For example, $K_{2,2,2}$ shown in Figure 2b is an ACM regular graph in $\mathcal{R}_{6,4}$ and $\lambda_2\left(K_{2,2,2}\right) = 4$. This result is consistent with Proposition 3.

We finally consider the case where $k = n - 3$. In this case, the following result is obtained.

Proposition 5. *Let n be an integer greater than or equal to six. If $n\mathrm{mod}3 = 0$ then any $G \in \mathcal{R}_{n,n-3}$ such that $\overline{G}$ is a disjoint union of cycles with three vertices is an ACM regular graph in $\mathcal{R}_{n,n-3}$ with $\lambda_2(G) = n - 3$. If $n\mathrm{mod}3 \neq 0$ and $n \geq 8$ then there exists a $G \in \mathcal{R}_{n,n-3}$ such that $\overline{G}$ is a disjoint union of cycles with either three or five vertices, and it is an ACM regular graph in $\mathcal{R}_{n,n-3}$ with $\lambda_2(G) = n - 2 + 2\cos(4\pi/5)$. If $n = 7$ then any $G \in \mathcal{R}_{7,4}$ such that $\overline{G}$ is a cycle with seven vertices is an ACM regular graph in $\mathcal{R}_{7,4}$ with $\lambda_2(G) = n - 2 + 2\cos(6\pi/7)$.*

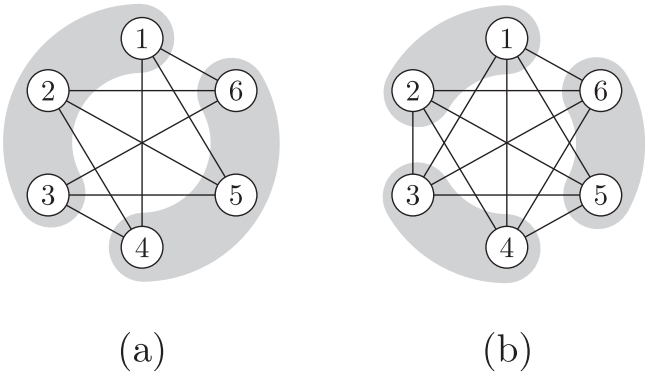

FIGURE 2 | ACM regular graphs: (a) complete bipartite graph $K_{3,3} \in \mathcal{R}_{6,3}$ and (b) complete tripartite graph $K_{2,2,2} \in \mathcal{R}_{6,4}$.

Proof. For any $G \in \mathcal{R}_{n,n-3}$, its complement graph $\overline{G}$ is a 2-regular graph. Hence $\overline{G}$ consists of either a disjoint union of multiple cycles or a single cycle. Let $C_{n_1}, C_{n_2}, \ldots, C_{n_\ell}$ be the connected components of $\overline{G}$, where ℓ is a positive integer and $\sum_{i=1}^{\ell} n_i = n$. Then the spectrum of the Laplacian matrix of $\overline{G}$ is given by $\cup_{i=1}^{\ell}\left\{\lambda_1\left(C_{n_i}\right) = 0, \lambda_2\left(C_{n_i}\right), \ldots, \lambda_{n_i}\left(C_{n_i}\right)\right\}$, and thus $\lambda_n(\overline{G}) = \max\left\{\lambda_{n_i}\left(C_{n_i}\right) \mid i = 1, 2, \ldots, n_\ell\right\}$. It follows from this fact and (2) that $\lambda_2(G)$ is maximized when $\lambda_n(\overline{G}) = \max\left\{\lambda_{n_i}\left(C_{n_i}\right) \mid i = 1, 2, \ldots, n_\ell\right\}$ is minimized. In addition, it is known that

$$\begin{aligned} \lambda_{n_i}\left(C_{n_i}\right) &= 2\left(1 - \cos\frac{2\pi\left\lceil\frac{n_i - 1}{2}\right\rceil}{n_i}\right) \\ &= \begin{cases} 4, & \text{if } n_i \bmod 2 = 0, \\ 2\left(1 - \cos\frac{\pi(n_i - 1)}{n_i}\right), & \text{otherwise}, \end{cases} \end{aligned}$$

where $\lceil r \rceil$ represents the smallest integer not less than r (see Ogiwara et al. [16] for example).

We first consider the case where $n\mathrm{mod}3 = 0$ and $n \geq 6$. In this case, there exists a $G \in \mathcal{R}_{n,n-3}$ such that $\overline{G}$ is a disjoint union of $n/3$ cycles with three vertices. The AC of such a graph G is given by

$$\lambda_2(G) = n - \lambda_n(\overline{G}) = n - \lambda_3\left(C_3\right) = n - 2\left(1 - \cos\frac{2\pi}{3}\right) = n - 3.$$

For any other $G \in \mathcal{R}_{n,n-3}$, at least one of the connected components of $\overline{G}$ is a cycle C_{n_i} with $n_i \geq 4$. If n_i is even then the AC of G is given by $\lambda_2(G) = n - 4$. If n_i is odd and $n_i \geq 5$ then the $\lambda_2(G) \leq n - 2(1 - \cos(4\pi/5)) < n - 3$. Therefore, any $G \in \mathcal{R}_{n,n-3}$ such that its complement graph $\overline{G}$ is a disjoint union of cycles with three vertices is an ACM regular graph in $\mathcal{R}_{n,n-3}$.

We next consider the case where $n\mathrm{mod}3 \neq 0$ and $n \geq 8$. In this case, there exists a $G \in \mathcal{R}_{n,n-3}$ such that $\overline{G}$ is a disjoint union of cycles with three or five vertices. To be more specific, when $n\mathrm{mod}3 = 2$ and $n \geq 8$, there is a $G \in \mathcal{R}_{n,n-3}$ such that $\overline{G}$ is a disjoint union of one cycle with five vertices and $(n-5)/3$ cycles with three vertices; when $n\mathrm{mod}3 = 1$ and $n \geq 10$, there is a $G \in \mathcal{R}_{n,n-3}$ such that $\overline{G}$ is a disjoint union of two cycles with five vertices and $(n-10)/3$ cycles with three vertices. The AC of such a graph G is given by

$$\lambda_2(G) = n - \lambda_n(\overline{G}) = n - \lambda_5\left(C_5\right) = n - 2\left(1 - \cos\frac{4\pi}{5}\right) \approx n - 3.618.$$

For any other $G \in \mathcal{R}_{n,n-3}$, at least one of the connected components of $\overline{G}$ is a cycle C_{n_i} with $n_i \notin \{3, 5\}$. Therefore, any graph $G \in \mathcal{R}_{n,n-3}$ such that its complement graph $\overline{G}$ is a disjoint union

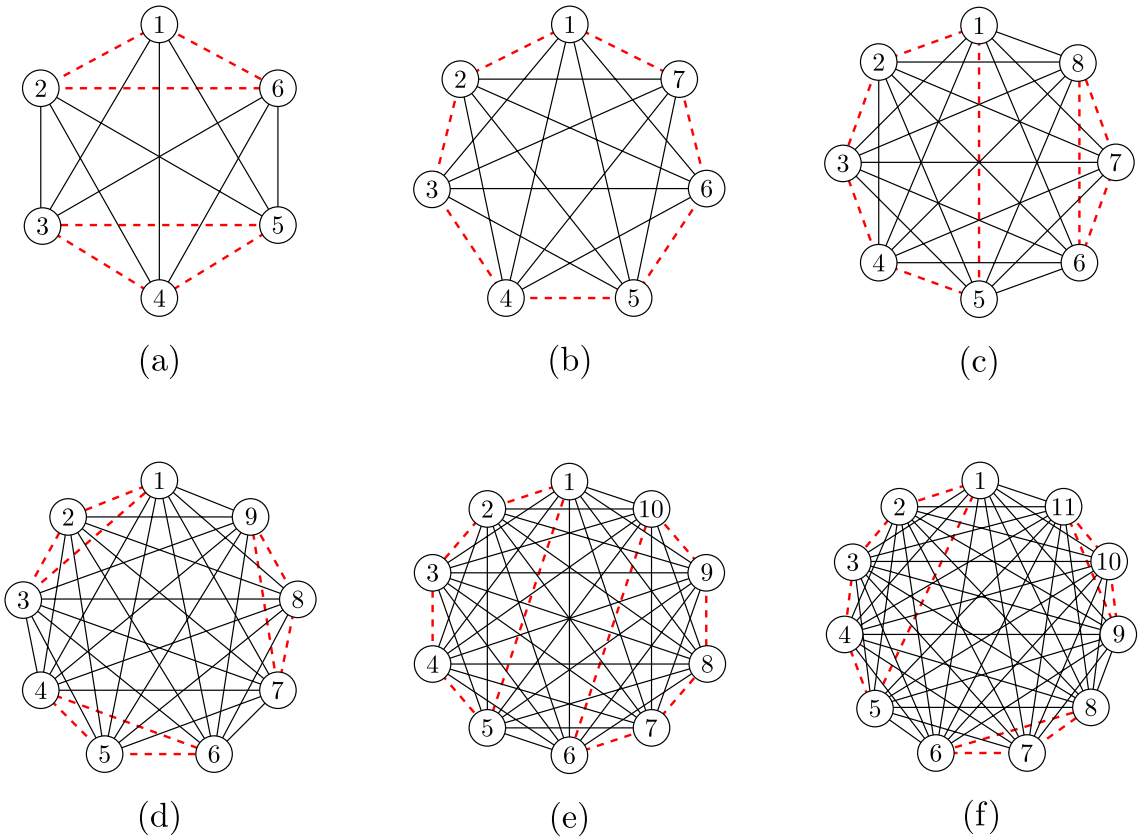

FIGURE 3 | ACM regular graphs in $\mathcal{R}_{n,n-3}$ with (a) $n = 6$, (b) $n = 7$, (c) $n = 8$, (d) $n = 9$, (e) $n = 10$ and (f) $n = 11$.

of cycles with three or five vertices is an ACM regular graph in $\mathcal{R}_{n,n-3}$.

We finally consider the case where $n = 7$. In this case, for any graph $G \in \mathcal{R}_{n,n-3}$, its complement graph $\overline{G}$ is a disjoint union of two cycles C_3 and C_4 or a cycle C_7. The AC of G is $n-4$ if $\overline{G}$ is a disjoint union of C_3 and C_4, and $n-2+2\cos(6\pi/7) > n-4$ otherwise. Therefore, any $G \in \mathcal{R}_{7,4}$ such that $\overline{G}$ is a cycle with seven vertices is an ACM regular graph in $\mathcal{R}_{7,4}$. □

ACM regular graphs in $\mathcal{R}_{n,n-3}$ with $n = 6, 7, \ldots, 11$ are shown in Figure 3. Figure 3a shows an ACM regular graph in $\mathcal{R}_{6,3}$, which is drawn with solid black lines, and its complement graph, which is drawn with dashed red lines. It is seen that the complement graph is a disjoint union of two cycles with three vertices. It is also seen that the ACM regular graph is a complete bipartite graph $K_{3,3}$. This result is consistent with Proposition 4. Figure 3b shows an ACM regular graph in $\mathcal{R}_{7,4}$ and its complement graph, which is a cycle with seven vertices. Figure 3c shows an ACM regular graph in $\mathcal{R}_{8,5}$ and its complement graph, which is a disjoint union of one cycle with five vertices and one cycle with three vertices. Figure 3d shows an ACM regular graph in $\mathcal{R}_{9,6}$ and its complement graph, which is a disjoint union of three cycles with three vertices. It is also seen that the ACM regular graph is a complete tripartite graph $K_{3,3,3}$. This result is consistent with Proposition 4. Figure 3e shows an ACM regular graph in $\mathcal{R}_{10,7}$ and its complement graph, which is a disjoint union of two cycles with five vertices. Figure 3f shows an ACM regular graph in $\mathcal{R}_{11,8}$ and its complement graph, which is a disjoint union of one cycle with five vertices and two cycles with three vertices.

4 | Depth-First Search Algorithms

4.1 | Simple Depth-First Search

We present in this section Depth-First Search (DFS) algorithms to find an ACM regular graph in $\mathcal{R}_{n,k}$ for a given pair (n, k) such that $3 \le k \le n-1$ and nk is even. Considering the discussion in the previous section, we hereafter focus our attention on the case where $n \ge 6$ and $3 \le k \le n-4$.

The basic idea behind our simple DFS algorithm is the same as the one for generalized Moore graphs [30, 31]. The algorithm starts with the initial graph $G_0 = (V, E_0)$ where $V = \{1, 2, \ldots, n\}$ and $E_0 = \{\{1,2\}, \{1,3\}, \ldots, \{1,k+1\}\}$, and the set $\hat{E} = \{\hat{e}_i\}_{i=1}^{M} = \overline{E}_0 \backslash \{\{1,k+2\}, \{1,k+3\}, \ldots, \{1,n\}\}$ where $\overline{E}_0$ is the edge set of the complement graph $\overline{G}_0$ of G_0 and M is the cardinality of $\hat{E}$. The algorithm then performs a DFS on the rooted binary tree in Figure 4 to find a subset E_+ of $\hat{E}$ such that $G_+ = (V, E_0 \cup E_+)$ is an ACM regular graph in $\mathcal{R}_{n,k}$. The root of the binary tree in Figure 4 represents the initial graph G_0. The transition from a node[1] at depth $i \in \{0, 1, \ldots, M-1\}$ to the left (right, resp.) child of the node corresponds to adopting (rejecting, resp.) $\hat{e}_{i+1}$. The symbols $+\hat{e}_i$ and $-\hat{e}_i$ in Figure 4 mean adoption and rejection, respectively, of the edge $\hat{e}_i \in \hat{E}$. Therefore, the paths from the root to the 2^M leaves represent the 2^M

ALGORITHM 1 | Depth-First Search.

Input: Order n and degree k
Output: An ACM regular graph in $\mathcal{R}_{n,k}$
1: Set $G^* \leftarrow (V, \emptyset)$, $\lambda_2^* \leftarrow 0$.
2: Set $G_0 \leftarrow (V, E_0)$ where $V = \{1, 2, \ldots, n\}$ and $E_0 = \{\{1,2\}, \{1,3\}, \ldots, \{1, k+1\}\}$.
3: Set $\widehat{E} = \{\hat{e}_i\}_{i=1}^{M} \leftarrow \overline{E}_0 \backslash \{\{1, k+2\}, \{1, k+3\}, \ldots, \{1, n\}\}$ where $\overline{E}_0$ is the edge set of $\overline{G}_0$.
4: Push $(0, G_0)$ into an empty stack S.
5: If S is empty then return G^* and stop. Otherwise pop $(i, G = (V, E))$ from S.
6: If G is not a k-regular graph then go to Step 10.
7: If $\lambda_2(G) = k$ then return G and stop.
8: If $\lambda_2^* < \lambda_2(G) < k$ then set $G^* \leftarrow G$ and $\lambda_2^* \leftarrow \lambda_2(G)$.
9: If $\lambda_2(G) \le \lambda_2^*$ then go to Step 5.
10: If $i < M$ and the maximum degree of $G' = (V, E \cup \{\hat{e}_{i+1}\})$ is not greater than k then push $(i+1, G')$ to S.
11: If $i < M$ and the minimum degree of $(V, E \cup \{\hat{e}_{i+2}, \hat{e}_{i+2}, \ldots, \hat{e}_M\})$ is not less than k then push $(i+1, G)$ to S.
12: Go to Step 5.

different subsets of $\widehat{E}$. Each node in the tree is identified by the depth of it and the graph G corresponding to the path from the root to it.

A formal description of the simple DFS algorithm is shown in Algorithm 1, where a stack is used to implement the DFS. Note that as soon as the algorithm finds a regular graph G with $\lambda_2(G) = k$, it returns G and stops (see Step 7). This is based on a well-known property of the AC that $\lambda_2(G)$ is not greater than the minimum degree of G [1]. Note also that Algorithm 1 does not push a node to the stack if the graph associated with the node has at least one vertex with degree greater than k (see Step 10). This is because there is clearly no k-regular graph in its descendants.

4.2 | Depth-First Search With Pruning

In order to avoid unnecessary search in Algorithm 1, we introduce four pruning techniques.

The first technique is based on the number of edges of the graph associated with each node of the binary tree (see Figure 5a). If the edge set E of the graph G associated with a node at depth i satisfies $|E| + M - i < nk/2$, we do not have to check its descendants because the maximum number of edges of the graphs associated with the descendants is less than $nk/2$, which means that there is no k-regular graph in the descendants.

The second one is based on the degree sequence of the graph associated with each node of the binary tree (see Figure 5b). Let $(k_{i_1}, k_{i_2}, \ldots, k_{i_n})$ be the nondecreasing sequence of n degrees of the graph G associated with a node at depth i. Then $(k - k_{i_1}, k - k_{i_2}, \ldots, k - k_{i_n})$ is the nonincreasing sequence of the missing degrees of G. If this sequence is not graphic, we do not have to check the descendants of the node because there is clearly no k-regular graph there.

The third one is based on the AC value of the graph associated with the left-most descendant of each node of the binary tree (see Figure 5c). It is well known that if two graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ satisfy $E_1 \subset E_2$ then $\lambda_2(G_1) \le \lambda_2(G_2)$ [1]. Taking this fact into account, we can say that, for each node of the binary tree, the AC value of the graph associated with its left-most descendant, which is denoted by $G_{\max}$, gives an upper bound for the AC values of the graphs associated with its descendants. If $\lambda_2(G_{\max})$ is less than or equal to λ_2^*, which is the maximum AC value obtained so far, then we do not have to check the descendants of the state.

The fourth one is based on the graph isomorphism (see Figure 5d). We choose the first and second members of $\widehat{E}$ as $\hat{e}_1 = \{2, 3\}$ and $\hat{e}_2 = \{2, 4\}$, respectively. In this case, if we exchange the vertex labels 3 and 4 in the edge sets $E_0 \cup \{\hat{e}_1\}$ and $\widehat{E} \backslash \{\hat{e}_1\}$ then we have $E_0 \cup \{\hat{e}_2\}$ and $\widehat{E} \backslash \{\hat{e}_2\}$, respectively. This means that the third node from the left at depth 2 of the rooted binary tree, which is represented by $(2, (VE_0 \cup \{\hat{e}_2\}))$, and its descendants have one-to-one correspondence with the second node from the left at depth 2, which is represented by $(2, (VE_0 \cup \{\hat{e}_1\}))$, and its descendants. The graph associated with a node in the former

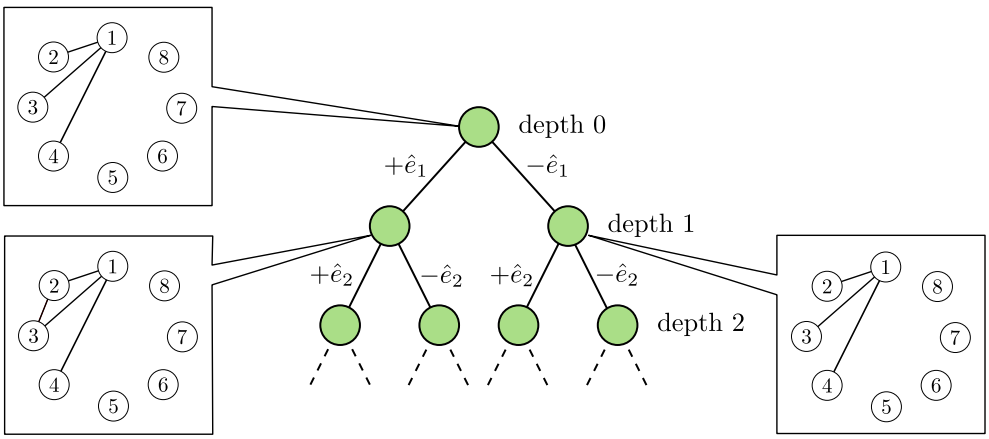

FIGURE 4 | Rooted binary tree representing edge selection process for the case where $(n, k) = (8, 3)$.

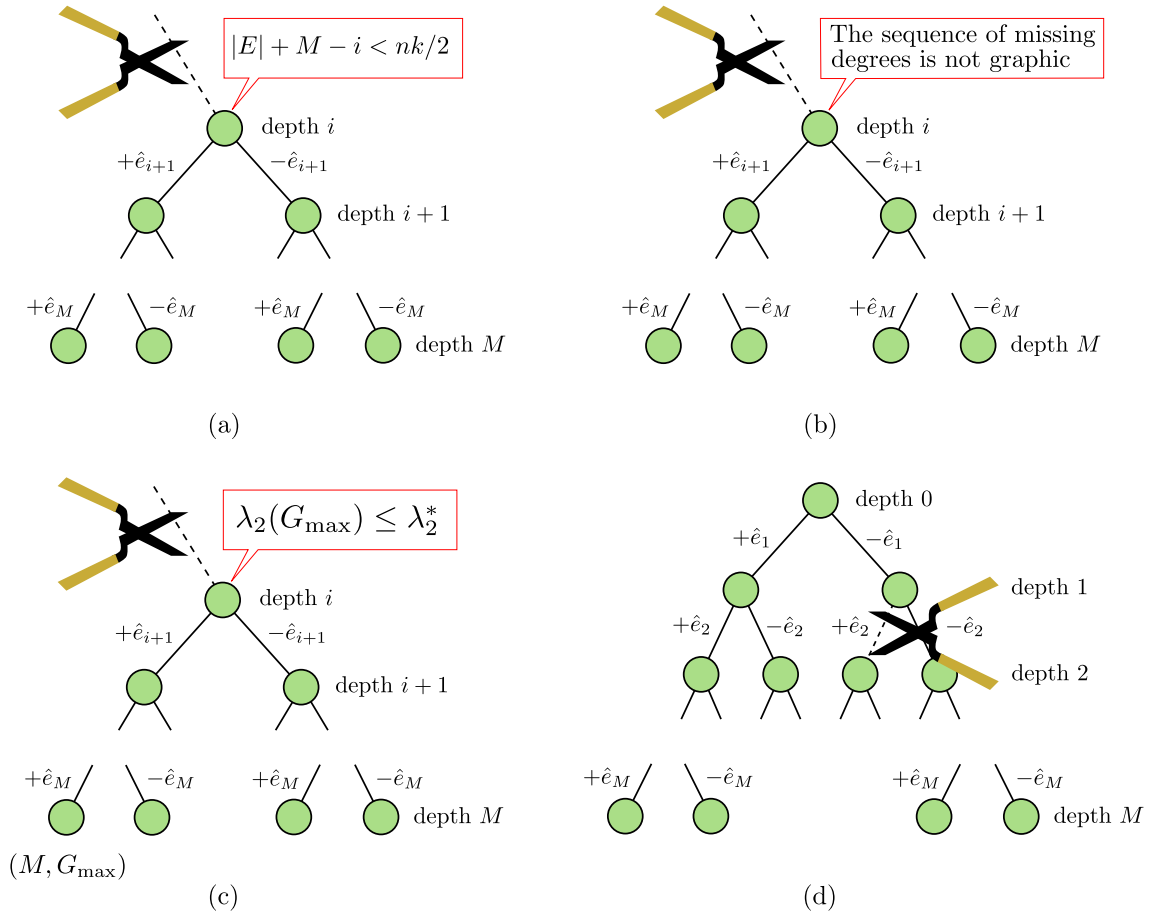

FIGURE 5 | Four pruning methods based on (a) the number of edges, (b) the sequence of missing degrees, (c) the AC of the graph corresponding to the left-most leaf, and (d) the graph isomorphism.

group is isomorphic to the one associated with the corresponding node in the latter group. Therefore, we do not have to check the third node from the left at depth 2 of the rooted binary tree and its descendants.

A formal description of a DFS algorithm with the above-mentioned pruning techniques is shown in Algorithm 2. The pruning methods based on (i) the number of edges, (ii) the sequence of missing degrees, (iii) the AC value of the graph corresponding to the left-most leaf, and (iv) the graph isomorphism are applied in Steps 10, 11, 12 and 4, respectively. The depth ratio r determines when the algorithm uses the pruning method based on the AC value of the graph corresponding to the left-most leaf. If $r = 0$ then this pruning method is used at every node of the binary tree. In this case, the execution time will be very long because the pruning condition $\lambda_2\left(\left(V, E \cup \left\{\hat{e}_{i+1}, \hat{e}_{i+2}, \ldots, \hat{e}_M\right\}\right)\right) < \lambda_2^*$ is not satisfied for nodes near the root. On the contrary, if $r \approx 1$ then the pruning has little or no effect because the pruning method is used at only nodes near leaves. Therefore, the value of r affects the execution time of the algorithm, and thus should be selected appropriately.

In the case where $k > n/2$, Algorithm 2 may take a long time to find an ACM regular graph. To avoid this situation, we use Algorithm 3 instead of Algorithm 2. Algorithm 3 searches for an $(n-k-1)$-regular graph G with the minimum $\lambda_n(G)$, because its complement graph $\overline{G}$ is a k-regular graph and $\lambda_2(\overline{G})$ is given by Equation (2).

4.3 | Example

In order to see how our pruning techniques work, we illustrate the behavior of Algorithm 2 for the case where $(n, k) = (6, 3)$ and $r = 0.6$ in Figure 6. We assume that $\hat{e}_1 = \{2,3\}$, $\hat{e}_2 = \{2,4\}$, $\hat{e}_3 = \{2,5\}$, $\hat{e}_4 = \{2,6\}$, $\hat{e}_5 = \{3,4\}$, $\hat{e}_6 = \{3,5\}$, $\hat{e}_7 = \{3,6\}$, $\hat{e}_8 = \{4,5\}$, $\hat{e}_9 = \{4,6\}$ and $\hat{e}_{10} = \{5,6\}$.

Figure 6a shows the transition of Algorithm 2 from the root of the binary tree to the node representing the graph $\left(V, E_0 \cup \left\{\hat{e}_1, \hat{e}_3\right\}\right)$ at depth 5. During the transition, pruning is performed three times. The first pruning is done at the left-most node representing the graph $\left(V, E_0 \cup \left\{\hat{e}_1, \hat{e}_2\right\}\right)$ at depth 2. Since the sequence (3,3,1,1,0,0) of the missing degrees of this graph is not graphic, there is no 3-regular graph in its descendants. The second one is done at the node representing the graph $\left(V, E_0 \cup \left\{\hat{e}_1, \hat{e}_3, \hat{e}_4\right\}\right)$ at depth 4. Since the degree of vertex 2 is four, there is no 3-regular graph in its descendants. Note that this type of pruning is not shown in Figure 5 but used in Line 13 of Algorithm 2. The third one is done at the node representing the graph $\left(V, E_0 \cup \left\{\hat{e}_1, \hat{e}_3, \hat{e}_5\right\}\right)$ at depth 5. Since the sequence (3,2,1,0,0,0)

ALGORITHM 2 | Depth-First Search With Pruning.

Input: Order n, degree k, and depth ratio $r \in [0,1)$

Output: An ACM regular graph in $\mathcal{R}_{n,k}$

1: Set $G^* \leftarrow (V, \emptyset)$, $\lambda_2^* \leftarrow 0$.
2: Set $G_0 \leftarrow (V, E_0)$ where $V = \{1, 2, \ldots, n\}$ and $E_0 = \{\{1,2\}, \{1,3\}, \ldots, \{1, k+1\}\}$.
3 Set $\widehat{E} = \{\widehat{e}_i\}_{i=1}^{M}$ where $\widehat{e}_1 = \{2,3\}$, $\widehat{e}_2 = \{2,4\}$ and the remaining $M-2$ edges are the members of $\overline{E}_0 \backslash \{\{1,k+2\}, \{1,k+3\}, \ldots, \{1,n\}, \{2,3\}, \{2,4\}\}$.
4: Push $(2, (VE_0))$, $(2, (VE_0 \cup \{\widehat{e}_1\}))$ and $(2, (V, E_0 \cup \{\widehat{e}_1, \widehat{e}_2\}))$ into an empty stack S.
5: If S is empty then return G^* and stop. Otherwise pop $(i, G = (V, E))$ from S.
6: If G is not a k-regular graph then go to Step 10.
7: If $\lambda_2(G) = k$ then return G and stop.
8: If $\lambda_2^* < \lambda_2(G) < k$ then set $G^* \leftarrow G$ and $\lambda_2^* \leftarrow \lambda_2(G)$.
9: If $\lambda_2(G) \le \lambda_2^*$ then go to Step 5.
10: If $|E| + M - i < nk/2$ then go to Step 5.
11: Check if the nonincreasing sequence $(k - k_{i_1}, k - k_{i_2}, \ldots, k - k_{i_n})$ of the missing degrees of G is graphic using the Havel-Hakimi algorithm. If it is not graphic, go to Step 5.
12: If $i \ge rM$ and $\lambda_2((V, E \cup \{\widehat{e}_{i+1}, \widehat{e}_{i+2}, \ldots, \widehat{e}_M\})) \le \lambda_2^*$ then go to Step 5.
13: If $i < M$ and the maximum degree of $G' = (V, E \cup \{\widehat{e}_{i+1}\})$ is not greater than k then push $(i+1, G')$ to S.
14: If $i < M$ and the minimum degree of $(V, E \cup \{\widehat{e}_{i+2}, \widehat{e}_{i+2}, \ldots, \widehat{e}_M\})$ is not less than k then push $(i+1, G)$ to S.
15: Go to Step 5.

ALGORITHM 3 | Depth-First Search With Pruning for $k > n/2$.

Input: Order n, Degree k, and depth ratio $r \in [0,1)$

Output: An ACM regular graph in $\mathcal{R}_{n,k}$

1: Set $G^* \leftarrow (V, \emptyset)$, $\lambda_2^* \leftarrow 0$.
2: Set $G_0 \leftarrow (V, E_0)$ where $V = \{1, 2, \ldots, n\}$ and $E_0 = \{\{1,2\}, \{1,3\}, \ldots, \{1, n-k\}\}$.
3: Set $\widehat{E} = \{\widehat{e}_i\}_{i=1}^{M}$ where $\widehat{e}_1 = \{2,3\}$, $\widehat{e}_2 = \{2,4\}$ and the remaining $M-2$ edges are the members of $\overline{E}_0 \backslash \{\{1,2\}, \{1,3\}, \ldots, \{1,n-k\}, \{2,3\}, \{2,4\}\}$.
4: Push $(2, (VE_0))$, $(2, (E_0 \cup \{\widehat{e}_1\}))$ and $(2, E_0 \cup \{\widehat{e}_1, \widehat{e}_2\})$ into an empty stack S.
5: If S is empty then return G^* and stop. Otherwise pop $(i, G = (V, E))$ from S.
6: If G is not an $(n-k-1)$-regular graph then go to Step 10.
7: If $n - \lambda_n(G) = k$ then return $\overline{G}$ and stop.
8: If $\lambda_2^* < n - \lambda_n(G) < k$ then set $G^* \leftarrow \overline{G}$ and $\lambda_2^* \leftarrow n - \lambda_n(G)$.
9: If $n - \lambda_n(G) \le \lambda_2^*$ then go to Step 5.
10: If $|E| + M - i < n(n-k-1)/2$ then go to Step 5.
11: Check if the nonincreasing sequence $(n-k-1-k_{i_1}, n-k-1-k_{i_2}, \ldots, n-k-1-k_{i_n})$ of the missing degrees of G is graphic using the Havel-Hakimi algorithm. If it is not graphic, go to Step 5.
12: If $i \ge rM$ and $n - \lambda_n(G) \le \lambda_2^*$ then go to Step 5.
13: If $i < M$ and the maximum degree of $G' = (V, E \cup \{\widehat{e}_{i+1}\})$ is not greater than $n-k-1$ then push $(i+1, G')$ to S.
14: If $i < M$ and the minimum degree of $(V, E \cup \{\widehat{e}_{i+2}, \widehat{e}_{i+2}, \ldots, \widehat{e}_M\})$ is not less than k then push $(i+1, G)$ to S.
15: Go to Step 5.

of the missing degrees of this graph is not graphic, there is no 3-regular graph in its descendants.

Figure 6b shows how Algorithm 2 searches for 3-regular graphs in the descendants of the node representing the graph $(V, E_0 \cup \{\widehat{e}_1, \widehat{e}_3\})$ at depth 5. During this search process, a 3-regular graph with the AC 2 is found at a leaf node, and pruning is performed five times. The first pruning is done at the node representing the graph $(V, E_0 \cup \{\widehat{e}_1, \widehat{e}_3, \widehat{e}_6\})$ at depth 6. Since the sequence (3,2,1,0,0,0) of the missing degrees of this graph is not graphic, there is no 3-regular graph in its descendants. The second one is done at the node representing the graph $(V, E_0 \cup \{\widehat{e}_1, \widehat{e}_3, \widehat{e}_7, \widehat{e}_8, \widehat{e}_9\})$ at depth 10. Since $|E| + M - i = 5 + 10 - 7 = 8 < 9 = nk/2$ holds, this graph is not a 3-regular graph. The third, fourth, and fifth pruning are done at the node representing the graph $(V, E_0 \cup \{\widehat{e}_1, \widehat{e}_3, \widehat{e}_7, \widehat{e}_8\})$ at depth 9, the node representing the graph $(V, E_0 \cup \{\widehat{e}_1, \widehat{e}_3, \widehat{e}_7\})$ at depth 8, and the node representing the graph $(V, E_0 \cup \{\widehat{e}_1, \widehat{e}_3\})$ at depth 7, respectively. Since these graphs satisfy $|E| + M - i < nk/2$, there is no 3-regular graph in their descendants.

Figure 6c shows how Algorithm 2 searches for 3-regular graphs in the descendants of the node representing the graph $(V, E_0 \cup \{\widehat{e}_1\})$ at depth 2. During this search process, no 3-regular graph is found, and pruning is performed five times. The first pruning is done at the node representing the graph $(V, E_0 \cup \{\widehat{e}_1, \widehat{e}_4, \widehat{e}_5\})$ at depth 5. Since the sequence (3,2,1,0,0,0) of the missing degrees of this graph is not graphic, there is no 3-regular graph in its descendants. The second one is done at the node representing the graph $(V, E_0 \cup \{\widehat{e}_1, \widehat{e}_4, \widehat{e}_6, \widehat{e}_7\})$ at depth 7. Since the degree of vertex 3 is four, there is no 3-regular graph in its descendants. The third one is done at the node representing the graph $(V, E_0 \cup \{\widehat{e}_1, \widehat{e}_4, \widehat{e}_6\})$ at depth 7, which is greater than $rM = 6$. Since the left-most leaf in its descendants represents the graph $(V, E_0 \cup \{\widehat{e}_1, \widehat{e}_4, \widehat{e}_6, \widehat{e}_8, \widehat{e}_9, \widehat{e}_{10}\})$ and the AC of

FIGURE 6 | Behavior of Algorithm 2 for the case where $(n, k) = (6, 3)$ and $r = 0.6$.

this graph is $\lambda_2^* = 2$, there is no graph with an AC higher than 2 in its descendants. The fourth one is done at the node representing the graph $\left(V, E_0 \cup \left\{\hat{e}_1, \hat{e}_4\right\}\right)$ at depth 6. Since the degree of vertex 5 in the graph represented by its left-most leaf is 2, there is no 3-regular graph in its descendants. Note that this type of pruning is not shown in Figure 5 but used in Line 14 of Algorithm 2. The fifth one is done at the node representing the graph $\left(V, E_0 \cup \left\{\hat{e}_1\right\}\right)$ at depth 4. Since the degree of vertex 2 in the graph represented by its left-most leaf is 2, there is no 3-regular graph in its descendants.

Figure 6d shows the transition of Algorithm 2 from the root of the binary tree to the node representing the graph $\left(V, E_0 \cup \left\{\hat{e}_3, \hat{e}_4, \hat{e}_5\right\}\right)$ at depth 5. During the transition, pruning based on the graph isomorphism is performed at the node representing the graph $\left(V, E_0 \cup \left\{\hat{e}_2\right\}\right)$.

Figure 6e,f show how Algorithm 2 searches for 3-regular graphs in the descendants of the node representing the graph $\left(V, E_0 \cup \left\{\hat{e}_3, \hat{e}_4, \hat{e}_5\right\}\right)$ at depth 5. During this search process, no 3-regular graph is found, and pruning is performed seven times. We do not go into the details, because the same techniques are used.

Figure 6g shows the transition of Algorithm 2 from the node representing the graph $\left(V, E_0 \cup \left\{\hat{e}_3, \hat{e}_4\right\}\right)$ at depth 4 to the node representing the graph $\left(V, E_0 \cup \left\{\hat{e}_3, \hat{e}_4, \hat{e}_6, \hat{e}_7, \hat{e}_8, \hat{e}_9\right\}\right)$ at depth 9. Since the latter is a 3-regular graph and its AC is 3, which is an upper bound, it is an ACM regular graph in $\mathcal{R}_{6,3}$. Note that it is certainly isomorphic to the graphs in Figures 2a and 3a. Algorithm 2 thus returns the graph $\left(V, E_0 \cup \left\{\hat{e}_3, \hat{e}_4, \hat{e}_6, \hat{e}_7, \hat{e}_8, \hat{e}_9\right\}\right)$ and stops the search.

5 | Evaluation of Proposed Algorithms

5.1 | Implementation

The authors implemented Algorithms 1–3 in Python language and executed on Computer 1 in Table 1. They also implemented parallel versions of Algorithms 2 and 3 using Message Passing Interface (MPI), and executed on the four computers in Table 1. Five processes run on each of Computers 1–3, and two processes run on Computer 4. One of the two processes on Computer 4 plays the role of a server, and the remaining 16 processes play the role of clients, as shown in Figure 7. The server first pushes 96

TABLE 1 | Specifications of computers used in experiments.

ID	CPU	OS	RAM	Python	CPU cores
1	AMD Ryzen 5 PRO 5650GE with Radeon Graphics	Ubuntu 22.04.4 LTS	14Gi	3.10.12	6
2	Intel Core i7-9700 CPU @ 3.00GHz	Ubuntu 22.04.4 LTS	15Gi	3.10.12	8
3	Intel Core i7-9700 CPU @ 3.00GHz	Ubuntu 22.04.4 LTS	15Gi	3.10.12	8
4	Intel Core i7-7500U CPU @ 2.70GHz	Ubuntu 22.04.4 LTS	7.7Gi	3.10.12	2

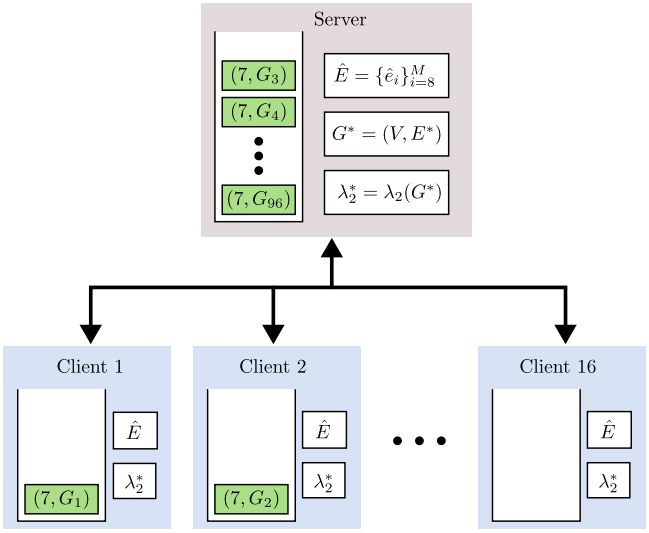

FIGURE 7 | Parallel implementation of the DFS with pruning.

nodes at depth 7 of the rooted binary tree, excluding the descendants of the third node from the left at depth 2, into an empty stack S, and set $G^* \leftarrow (V, \emptyset)$ and $\lambda_2^* \leftarrow 0$. The server then pops a node (i, G) from S as requested by a client, and sends the graph G together with the edge list $\hat{E} = \{\hat{e}_i\}_{i=8}^{M}$ and the value of λ_2^* to the client. The client performs the DFS on the rooted binary tree with the root representing $(0, G)$ like Algorithm 2 or 3 using its own stack. If a client finds a regular graph with a larger AC value than λ_2^* then it sends the AC value and the graph to the server, and the server updates the value of λ_2^* and the corresponding regular graph G^*.

5.2 | Effectiveness of Pruning and Parallel Implementation

In order to evaluate the effectiveness of the proposed pruning and the parallel implementation, the authors ran the following three Python programs for some values of (n, k) and measured execution time: (i) the simple DFS in Algorithm 1, (ii) the DFS with pruning in Algorithms 2 and 3, and (iii) the parallel implementation of the DFS with pruning. All programs were configured to forcibly terminate when the execution time reaches 48 h (172,800 s), even if the search process is still ongoing. In the second and third programs, the value of the depth ratio r was set to 0.6. The authors confirmed through preliminary experiments that this value minimizes the execution time of the DFS with pruning (see Table 2).

The results of the experiments are shown in Table 3. The simple DFS algorithm found an ACM regular graph when $n \leq 10$, but could not when $n = 11$ and $n = 12$. The DFS with pruning found an ACM regular graph when $n \leq 11$, and the execution time is significantly shorter than that of the simple DFS algorithm. To be more precise, the ratio of the execution time of the DFS with pruning to that of the simple DFS is less than 0.025 in all cases, with the minimum value being 0.00062 when $(n, k) = (10, 6)$. This indicates that the pruning techniques proposed in this paper work very effectively. However, when $n = 12$, even this algorithm could not complete the search process within 48 h. The execution time of the parallel implementation of the DFS with pruning is longer than that of the serial implementation when $(n, k) = (8, 3)$ and $(n, k) = (9, 4)$ due to overhead in parallel processing, but is significantly shorter in other cases. To be more precise, the ratio of the execution time of the parallel implementation of the DFS with pruning to that of the single process is less than 0.3 for $(n, k) = (10, 3), (10, 4), (10, 6), (11, 4)$ and $(11, 6)$. In particular, the execution time of the parallel implementation is only 4% of that of the single process one when $(n, k) = (11, 6)$. This indicates that the parallel processing is very useful for speeding up the DFS algorithm with pruning.

5.3 | Characterization of Obtained ACM Regular Graphs

The AC values of the ACM regular graphs in $\mathcal{R}_{n,k}$ for some values of (n, k) obtained using the Python programs are shown in Table 4. The expression "(at least x)" for the cases where $(n, k) = (12, 4)$ and $(n, k) = (12, 5)$ means that a regular graph with the AC value x was found but it is not known whether x is the maximum or not because the search did not complete within 48 h. Note that the programs were run even in the cases where the AC values of ACM regular graphs are known from the theoretical analysis in Propositions 1–5. It is clear from the table that the AC values obtained using the programs coincide with the theoretical results.

The ACM regular graphs obtained using Algorithm 2 for $(n, k) = (8, 3), (9, 4), (10, 3), (10, 4), (11, 4)$ and $(12, 4)$ are shown in Figure 8, and those obtained using Algorithm 3 for $(n, k) = (10, 6), (11, 6)$ and $(12, 7)$ are shown in Figure 9. Note that the vertices of each graph are relabeled so that the structure of the graph can be easily seen. This is the reason why vertex 1 of the graph in Figure 8a, for example, is not adjacent to vertices 2, 3 and 4 while the initial graph G_0 in Algorithms 2 and 3 always contains the edges $\{1, 2\}$, $\{1, 3\}$ and $\{1, 4\}$. We see from Figures 8 and 9 that all the graphs obtained by Algorithms 2 and 3 are multipartite graphs. These results indicate that multipartite graphs are closely related to ACM regular graphs.

TABLE 2 | Execution time (in seconds) of the DFS with pruning for various values of r.

	(n, k)				
r	**(8, 3)**	**(9, 4)**	**(10, 3)**	**(10, 4)**	**(10, 6)**
0.3	0.0888	4.861	63.491	314.805	134.032
0.4	0.0790	3.213	39.330	214.044	86.374
0.5	0.0561	2.182	27.387	112.813	53.913
0.6	**0.0553**	**1.727**	**11.630**	**99.991**	**53.788**
0.7	0.0652	2.747	16.527	154.828	100.430
0.8	0.0798	4.205	56.849	301.471	162.172
0.9	0.0870	5.226	70.897	397.421	179.428

Note: Bold values means that it is the smallest among the values in the same column.

TABLE 3 | Execution time (in seconds) of the simple DFS, the DFS with pruning, and a parallel implementation of the DFS with pruning.

	(n, k)								
	(8, 3)	**(9, 4)**	**(10, 3)**	**(10, 4)**	**(10, 6)**	**(11, 4)**	**(11, 6)**	**(12, 3)**	**(12, 7)**
Algorithm									
Simple (T_s)	2.358	374.851	8996.812	61493.487	77963.179	—	—	—	—
Pruning (T_p)	**0.058**	**1.800**	12.168	102.982	48.679	7980.596	26271.688	—	—
Multiprocess (T_m)	2.154	2.447	**3.526**	**12.822**	**6.932**	**834.568**	**1099.969**	**3844.446**	**90037.491**
Ratio of execution time									
T_p/T_s	0.02460	0.00480	0.00135	0.00167	0.00062	—	—	—	—
T_m/T_p	37.13793	1.35944	0.28978	0.12451	0.14240	0.10457	0.04187	—	—

Note: Bold value means that it is the smallest among the values of the three algorithms: simple, pruning, and multiprocess for each value of (n, k).

TABLE 4 | AC values of ACM regular graphs.

n	$k = 3$	$k = 4$	$k = 5$	$k = 6$	$k = 7$	$k = 8$	$k = 9$	$k = 10$	$k = 11$
6	3	4	6	—	—	—	—	—	—
7	—	3.198	—	7	—	—	—	—	—
8	2[a]	4	4.382	6	8	—	—	—	—
9	—	3[a]	—	6	—	9	—	—	—
10	2[a]	3[a]	5	5	6.382	8	10	—	—
11	—	2.602[a]	—	5[a]	—	7.382	—	11	—
12	1.468[a]	(at least 3)[a]	(at least 4)[a]	6	6[a]	8	9	10	12

Note: The expression "(at least x)" means that a regular graph with the AC value x was found but it is not known whether x is the maximum or not because the search did not complete within 48 h.
[a]indicates that the value cannot be obtained from Propositions 1–5.

Although the Python program could not complete the DFS process in 48 h when $(n, k) = (12, 4)$ and $(12, 5)$, it returned k-regular graphs with the highest AC value among all the k-regular graphs it examined, which are shown in Figure 10. As in Figures 8 and 9, the vertices of each graph are relabeled so that the structure of the graph can be easily seen. It is clear that the two graphs in Figure 10 are multipartite graphs. Since the AC values of these graphs are 3 and 4, we can say that the AC values of the ACM regular graphs in $\mathcal{R}_{12,4}$ and $\mathcal{R}_{12,5}$ are at least 3 and 4, respectively. As we have observed that multipartite graphs are closely related to ACM regular graphs, the graphs in Figure 10 may be ACM regular graphs in $\mathcal{R}_{12,4}$ and $\mathcal{R}_{12,5}$.

6 | Conclusions

In this paper, we considered the problem of finding a k-regular graph with n vertices that has the maximum AC for a given pair

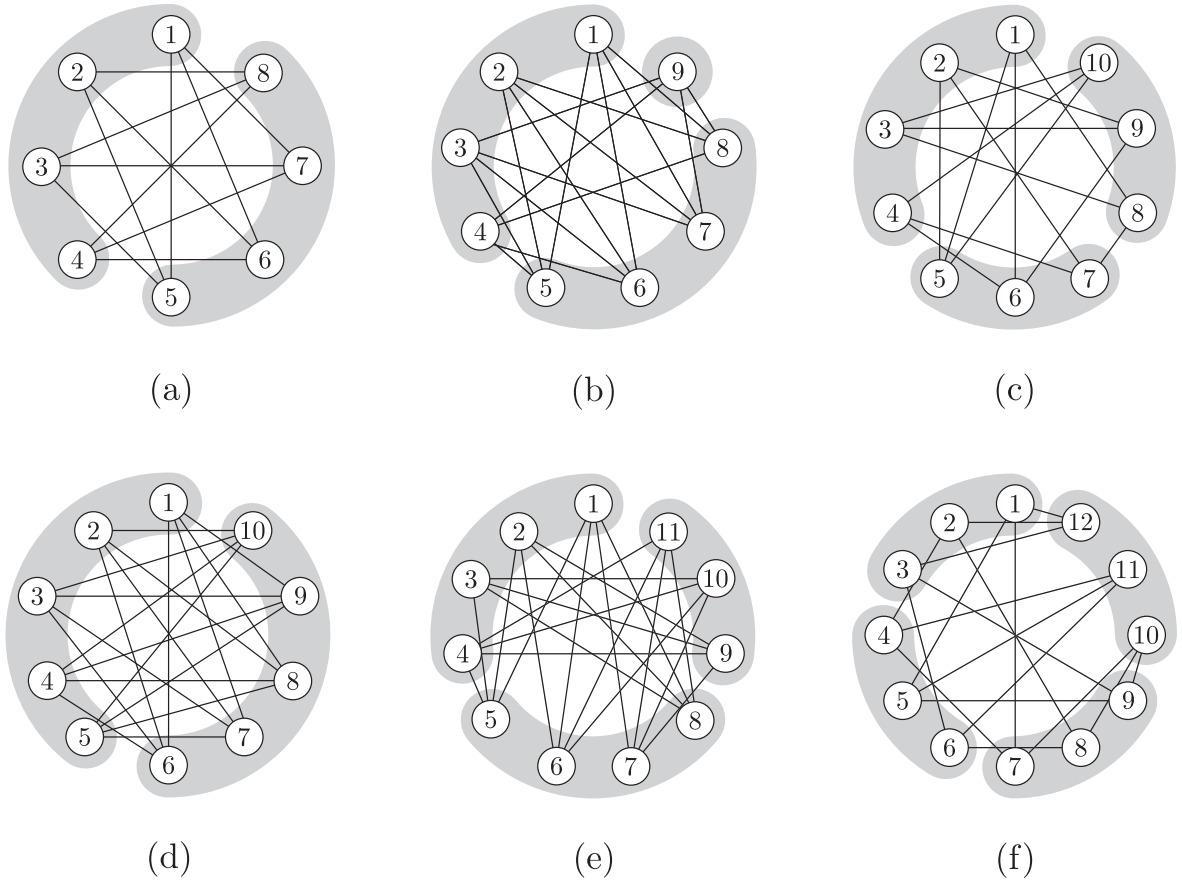

FIGURE 8 | ACM regular graphs in $\mathcal{R}_{n,k}$ obtained using Algorithm 2 for (a) $(n, k) = (8, 3)$, (b) $(n, k) = (9, 4)$, (c) $(n, k) = (10, 3)$, (d) $(n, k) = (10, 4)$, (e) $(n, k) = (11, 4)$ and (f) $(n, k) = (12, 3)$.

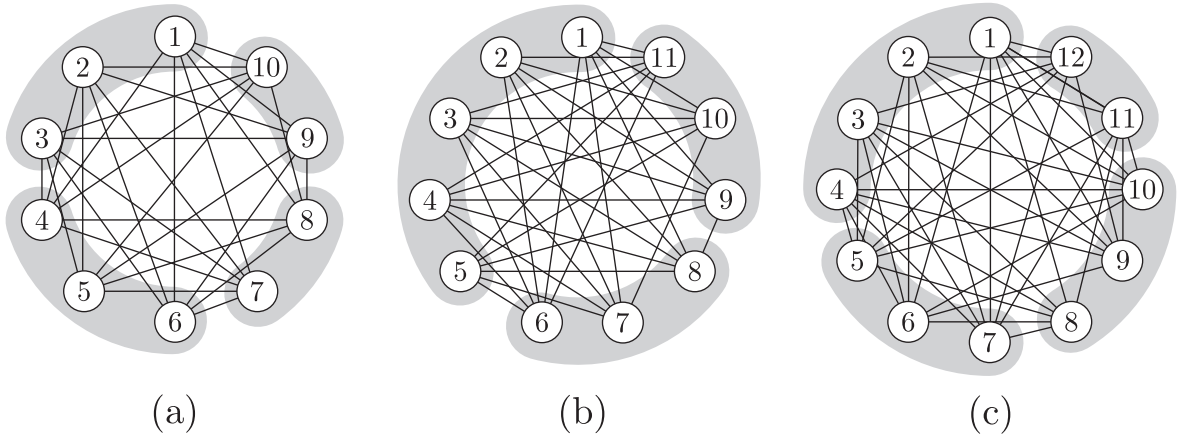

FIGURE 9 | ACM regular graphs in $\mathcal{R}_{n,k}$ obtained using Algorithm 3 for (a) $(n, k) = (10, 6)$, (b) $(n, k) = (11, 6)$ and (c) $(n, k) = (12.7)$.

(n, k). We first focused our attention on some special cases where $k \in \{2, n-3, n-2, n-1\}$ or $k = n(p-1)/p$ with p being a divisor of n, and derived solutions through theoretical analysis. We next developed some depth-first search algorithms for solving this problem, and obtained solutions for the cases where $n \leq 12$. The developed algorithms make use of some pruning techniques to reduce the search space, and parallel implementation to achieve reasonable speed-up. An important property of the solutions obtained by the proposed algorithms is that all of them can be considered as multipartite graphs.

One of the future challenges is to determine whether the graphs shown in Figure 10 are solutions for the cases where $(n, k) = (12, 4)$ and $(12, 5)$. Other challenges are to understand the relationship between the multipartite graphs and the solutions of the problem through theoretical analysis, and to develop more

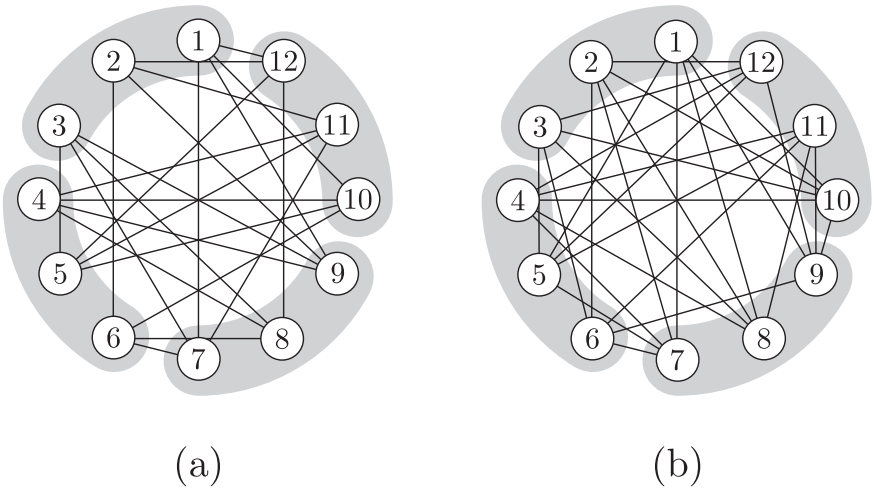

FIGURE 10 | Possible ACM regular graphs in $\mathcal{R}_{n,k}$ obtained using Algorithm 2 for (a) $(n, k) = (12, 4)$ and (b) $(n, k) = (12, 5)$.

efficient pruning techniques by making use of the properties of the AC.

Acknowledgments

The authors would like to thank anonymous reviewers for their valuable comments and suggestions to improve the quality of the paper.

Conflicts of Interest

The authors declare no conflicts of interest.

Data Availability Statement

Data sharing not applicable to this article as no datasets were generated or analyzed during the current study.

Endnotes

[1] We use the term "node" in order to distinguish from vertices of graphs.

References

1. M. Fiedler, "Algebraic Connectivity of Graphs," *Czechoslovak Mathematical Journal* 23, no. 98 (1973): 298–305.

2. M. Fiedler, "Laplacian of Graphs and Algebraic Connectivity," *Combinatorics and Graph Theory* 25 (1987): 57–70.

3. d N M M. Abreu, "Old and New Results on Algebraic Connectivity of Graphs," *Linear Algebra and Its Applications* 423, no. 1 (2007): 53–73.

4. P. Wei, L. Chen, and D. Sun, "Algebraic Connectivity Maximization of an Air Transportation Network: The Flight Routes' Addition/Deletion Problem," *Transportation Research Part E: Logistics and Transportation Review* 61 (2014): 13–27.

5. P. Wei, G. Spiers, and D. Sun, "Algebraic Connectivity Maximization for Air Transportation Networks," *IEEE Transactions on Intelligent Transportation Systems* 15, no. 2 (2014): 685–698.

6. K. F. Cheung and M. G. Bell, "Improving Connectivity of Compromised Digital Networks via Algebraic Connectivity Maximisation," *European Journal of Operational Research* 294, no. 1 (2021): 353–364.

7. A. Tavasoli, H. Shakeri, E. Ardjmand, and S. Rahman, "The Art of Interconnections: Achieving Maximum Algebraic Connectivity in Multilayer Networks," *Networking Science* 12, no. 3 (2024): 261–288.

8. N. Somisetty, H. Nagarajan, and S. Darbha, "Optimal Robust Network Design: Formulations and Algorithms for Maximizing Algebraic Connectivity," *IEEE Transactions on Control of Network Systems* 12, no. 1 (2025): 918–929.

9. S. Boyd, P. Diaconis, and L. Xiao, "Fastest Mixing Markov Chain on a Graph," *SIAM Review* 46, no. 4 (2004): 667–689.

10. J. Sun, S. Boyd, L. Xiao, and P. Diaconis, "The Fastest Mixing Markov Process on a Graph and a Connection to a Maximum Variance Unfolding Problem," *SIAM Review* 48, no. 4 (2006): 681–699.

11. R. Olfati-Saber, J. A. Fax, and R. M. Murray, "Consensus and Cooperation in Networked Multi-Agent Systems," *Proceedings of the IEEE* 95, no. 1 (2007): 215–233.

12. Z. Li, Z. Duan, G. Chen, and L. Huang, "Consensus of Multiagent Systems and Synchronization of Complex Networks: A Unified Viewpoint," *IEEE Transactions on Circuits and Systems I: Regular Papers* 57, no. 1 (2009): 213–224.

13. M. S. Sarafraz and M. S. Tavazoei, "A Unified Optimization-Based Framework to Adjust Consensus Convergence Rate and Optimize the Network Topology in Uncertain Multi-Agent Systems," *IEEE/CAA Journal of Automatica Sinica* 8, no. 9 (2021): 1539–1548.

14. Y. Kim and M. Mesbahi, "On Maximizing the Second Smallest Eigenvalue of a State-Dependent Graph Laplacian," *IEEE Transactions on Automatic Control* 51, no. 1 (2006): 116–120.

15. R. Dai and M. Mesbahi, "Optimal Topology Design for Dynamic Networks," in *Proceedings of the 50th IEEE Conference on Decision and Control and European Control Conference* (IEEE, 2011), 1280–1285.

16. K. Ogiwara, T. Fukami, and N. Takahashi, "Maximizing Algebraic Connectivity in the Space of Graphs With a Fixed Number of Vertices and Edges," *IEEE Transactions on Control of Network Systems* 4, no. 2 (2017): 359–368.

17. R. Ishii and N. Takahashi, "Extensions of a Theorem on Algebraic Connectivity Maximizing Graphs," in *Proceedings of the 2016 International Symposium on Nonlinear Theory and Its Applications* (IEICE, 2016), 598–601.

18. G. Li, Z. F. Hao, H. Huang, and H. Wei, "Maximizing Algebraic Connectivity via Minimum Degree and Maximum Distance," *IEEE Access* 6 (2018): 41249–41255.

19. M. Koibuchi, I. Fujiwara, S. Hirasawa, et al., "Graph Golf: The Order/Degree Problem Competition," 2015–2021, https://research.nii.ac.jp/graphgolf/.

20. J. Friedman, "Some Geometric Aspects of Graphs and Their Eigenfunctions," *Duke Mathematical Journal* 69, no. 3 (1993): 487–525.

21. A. Nilli, "Tight Estimates for Eigenvalues of Regular Graphs," *Electronic Journal of Combinatorics* 11, no. 1 (2004): N9.

22. R. Olfati-Saber, "Algebraic Connectivity Ratio of Ramanujan Graphs," in *Proceedings of the 2007 American Control Conference* (IEEE, 2007), 4619–4624.

23. K. Shahbaz, M. N. Belur, and A. Ganesh, "Algebraic Connectivity: Local and Global Maximizer Graphs," *IEEE Transactions on Network Science and Engineering* 10, no. 3 (2023): 1636–1647.

24. G. Exoo, T. Kolokolnikov, J. Janssen, and T. Salamon, "Attainable Bounds for Algebraic Connectivity and Maximally Connected Regular Graphs," *Journal of Graph Theory* 107, no. 3 (2024): 522–549.

25. T. Kolokolnikov, "It Is Better to Be Semi-Regular When You Have a Low Degree," *Entropy* 26, no. 12 (2024): 1014.

26. K. Sato and S. I. Nakano, "Enumeration of Graphs With a Specified Degree Sequence," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Science (Japanese Edition)* 91, no. 7 (2008): 716–725.

27. H. Kim, Z. Toroczkai, P. L. Erdős, I. Miklós, and L. A. Székely, "Degree-Based Graph Construction," *Journal of Physics A: Mathematical and Theoretical* 42, no. 39 (2009): 392001.

28. M. Kurahashi, N. Salaani, T. Migita, and N. Takahashi, "Depth-First Search Algorithms for Algebraic Connectivity Maximizing Regular Graphs," in *Proceedings of the 2024 Twelfth International Symposium on Computing and Networking* (IEEE, 2024), 156–161.

29. D. B. West, *Introduction to Graph Theory, Second Edition* (Prentice hall Upper Saddle River, 2001).

30. Y. Satotani and N. Takahashi, "Depth-First Search Algorithms for Finding a Generalized Moore Graph," in *Proceedings of the 2018 IEEE Region 10 Conference* (IEEE, 2018), 832–837.

31. T. Hirayama, T. Migita, and N. Takahashi, "A Faster Algorithm to Search for Generalized Moore Graphs," in *Proceedings of the 2022 IEEE Region 10 Conference* (IEEE, 2022), 1–6.