What do those graphs look like?

Kun Wang, Texas A&M University

Steven Ning, TAMU PReMa & Friendswood High School

TAMU Math Circle

- Discovery Session (1–3pm)
- Problem-solving Session (3–4pm)

PReMa (Program for Research in Mathematics)

is a free online research program focusing on research for high school students.

This year, we are fortunate to receive support from the MAA Dolciani Mathematics Enrichment Grants (DMEG), which is a great help for our students.

AMBARZUMIAN-TYPE MIXED INVERSE SPECTRAL PROBLEMS FOR JACOBI MATRICES

ETHAN LUO, STEVEN NING, TARUN RAPAKA, AND JOYCE ZHENG

ABSTRACT

We investigate Ambarzumian-type mixed inverse spectral problems for Jacobi matrices. Specifically, we examine whether the Jacobi matrix can be uniquely determined by knowing all but the first m diagonal entries and a set of m consecutive eigenvalues.

This paper was submitted to Pi Mu Epsilon Journal and is under minor revision now.

Problem

We are interested in a simple graph G with the following conditions:

i) There is no triangle in this graph

ii) For any two vertices that are not adjacent then there are exactly two other vertices connecting to both of these two vertices

What do those graphs look like?

Group Activities of Searching Feasible Graphs (2D or 3D)

- Using pencil and paper
- Using playdough and toothpicks, PVC, strings
- Debug from a list of certain graphs

More to go…

Discover more features

Do programming

Use advanced math techniques to speed up the programming process

Discovering the Features

To make searching for the next graph easier, we would like to discover some features.

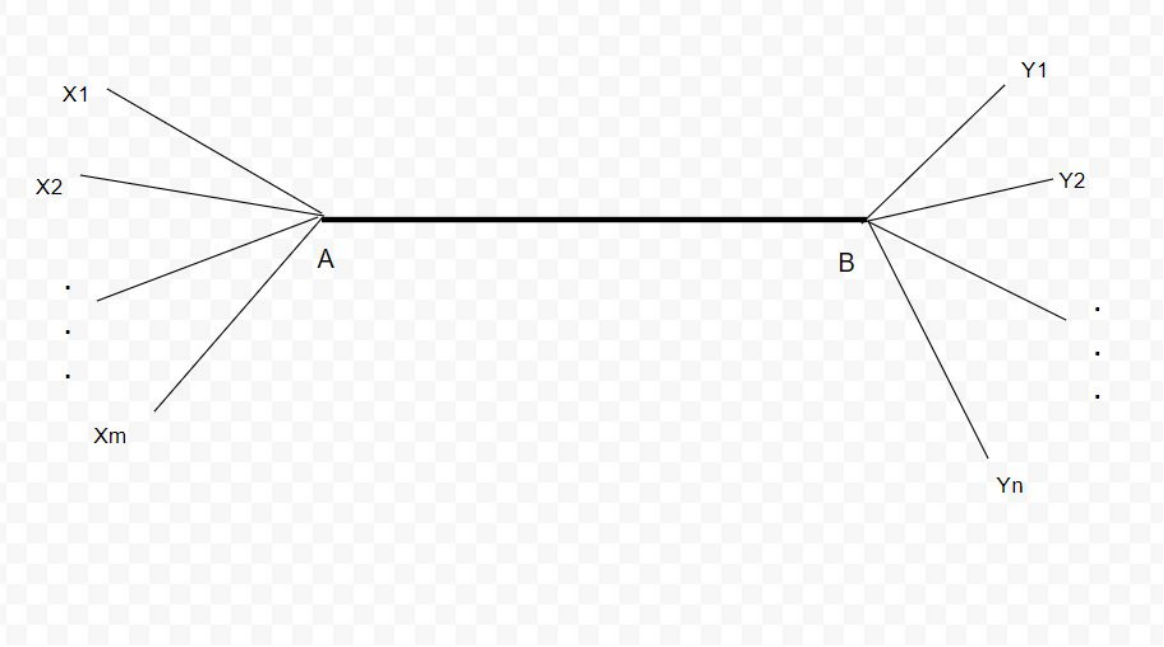
Theorem 1: Such graph is regular (every vertex has the same degree).

Theorem 2: There is some certain relationship between the degree k and the total number of vertices n for the graphs satisfying the conditions.

I. Proof of Regularity

It's easy to see that G is a connected graph. To show G is regular, it is enough to prove any two adjacent vertices of G have the same degree.

To prove any two adjacent vertices has the same degree, we will define a bijective map between their neighbors.



Let's name the adjacent vertices of A (other than B) to be $X_1, X_2, ..., X_m$ and name the adjacent vertices of B (other than A) to be $Y_1, Y_2, ..., Y_n$. Set $X = \{X_1, X_2, ..., X_m\}$ and set $Y = \{Y_1, Y_2, ..., Y_n\}$.

Define a map $f : X \to Y$. Because of Condition i), B and X_i are not adjacent and by Condition ii), B and X_i have two common neighbors. One of the two common neighbors is A, the other must be some element Y_j in Y. So $f(X_i) = Y_j$.

Then we prove f is injective. By Condition i), A and Y_i are nonadjacent. So, they should have two common neighbors. Assume $X_i \to Y_i$ and $X_j \to Y_i$. That means Y_i would be adjacent to X_i, X_j, B , which contradicts to Condition ii). So the map is injective. Hence $m \leq n$.

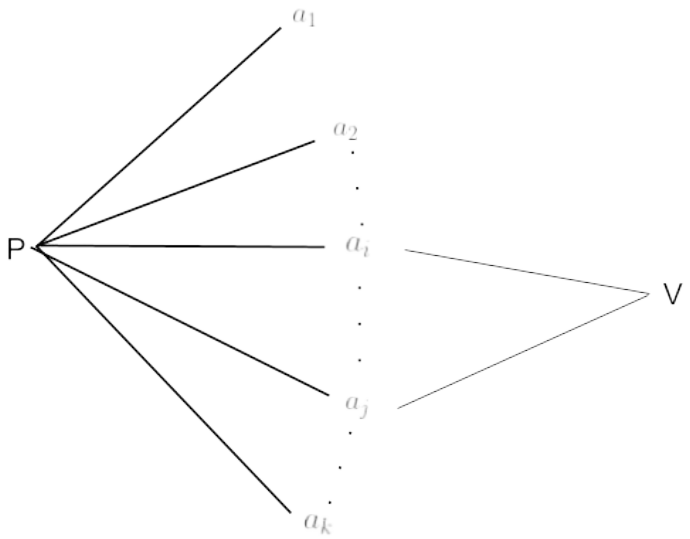
Because of the arbitrariness of A and B, we can set an injective map $f' : Y \to X$ similarly. Hence $n \leq m$.

The map f is a bijection, hence m=n. A has degree m+1, B has degree n+1, so A and B have same degree. We have proved G is connected, so G is regular.

II. Finding the Relation between Degree and Vertices

Theorem 2: For the graph G with n vertices from Theorem 1, set the degree of G to be k. Then $n = \frac{k(k+1)}{2} + 1$.

Solution: For a vertex P, there are exactly k neighbors, namely $a_1, a_2, ..., a_k$. Consider another vertex V that is not adjacent to P. By condition ii), V should connect to any two vertices from $\{a_1, a_2, ..., a_k\}$. There are as many as $\binom{k}{2}$ choices for the common neighbors of V.



Next we prove that V is **unique** for connecting to the certain pairs a_i and a_j. If there is another vertex V' also connected to a_i and a_j, a_i would have 3 common neighbors to a_j: P, V, V'. That contradicts condition ii) since a_i and a_j are nonadjacent. (If a_i and a_j are adjacent, then (V, a_i, a_j) form a triangle.)

Hence, V is unique, which means there are $\binom{k}{2} = \frac{k(k-1)}{2}$ such nonadjacent vertices.

There are k vertices adjacent to P and $\binom{k}{2}$ vertices not adjacent to P. Therefore, total vertices $n = k + 1 + \binom{k}{2}$.

Simplifies into $n = k + 1 + \frac{k(k-1)}{2} = \frac{k(k+1)}{2} + 1$

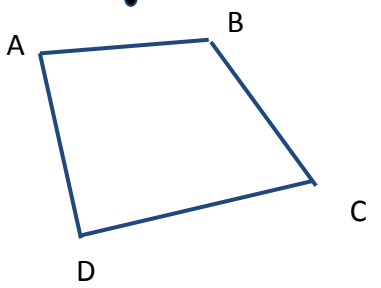
Applying the Features

A few trivial examples of this type of graph include:

- For k=0, n=1, which makes a complete graph of 1 vertex, K_1

- For k=1, n=2, which makes a complete graph of 2 vertices, K_2

- For k=2, n=4, which makes a cycle of 4 vertices, C_4



Continuing to use the pattern between degree and number of vertices, the next possible graph would be **3-regular, 7-vertex**.

Then **4-regular, 11-vertex.**

Then **5-regular, 16-vertex etc.**

Utilizing Adjacency Matrix

Definition: For a graph G with vertex set $U = \{u_1, \ldots, u_n\}$, the **adjacency matrix** is a square $n \times n$ matrix A such that its element A_{ij} is one when there is an edge from vertex u_i to vertex u_j, and zero when there is no edge.

Remark: For a simple graph of n vertices G, its adjacency matrix A is nxn with entries either 0 or 1(unweighted), symmetric(undirected), and its diagonal entries are pure 0's(no loops).

Proposition: If A is the adjacency matrix of a graph G, then the element (i, j) of A^n gives the number of paths of length n from vertex i to vertex j.

Algorithm Analysis

- G is k-regular, i.e. any vertex has k neighbours, hence k two-step paths to itself. That means all the diagonal entries of A^2 is k.

- Condition i) states “there are no triangles in the graph”, i.e. there are no 3-step paths to itself. So, the diagonal entries of A^3 must be pure 0’s.

- Condition ii) states “for any two vertices that are not adjacent then there are exactly two other vertices connecting to both of these two vertices”. That means if $A_{mn} = 0$,then $A^2_{mn} = 2$.

Programming the Search

jupyter matrix Last Checkpoint: last month

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

[11]:

```
s = 7
d = 3

#setting up the adjacency matrix A with sxs
A = np.zeros((s,s))
for k in range(0,s):
        for t in range(0,s):
            A[k][t] = random.randint(0,1)
            A[t][k] = A[k][t]
            A[t][t]=0
print("The adjacency matrix A is")
print(A)
A2 = np.dot(A,A)
A3 = np.dot(A2,A)
print("A^2 is")
print(A2)
print("A^3 is")
print( A3)

#checking the regularity
for k in range (0,s):
    if A2[k][k] != d:
        print("A is not regular")
        sys.exit()
    else:
        continue

#checking Condition #1
    for k in range (0,s):
        if A3[k][k] != 0:
            print("A does not follow condition 1")
            sys.exit()
    else:
        continue

#checking Condition #2
    for i in range (0,s):
        for j in range (0,s):
            if A[i][j] == 0:
                if A2[i][j]!=2:
                    print ("A does not follow condition 2")
                    sys.exit()
                else:
                    continue

print("A satifies both conditions")
```

A result from execution:

```
The adjacency matrix A is
[[0. 1. 0. 1. 0. 0. 1.]
 [1. 0. 0. 1. 1. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0.]
 [1. 1. 0. 0. 0. 1. 0.]
 [0. 1. 0. 0. 0. 0. 0.]
 [0. 1. 0. 1. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0.]]
A^2 is
[[3. 1. 0. 1. 1. 2. 0.]
 [1. 4. 0. 2. 0. 1. 1.]
 [0. 0. 0. 0. 0. 0. 0.]
 [1. 2. 0. 3. 1. 1. 1.]
 [1. 0. 0. 1. 1. 1. 0.]
 [2. 1. 0. 1. 1. 2. 0.]
 [0. 1. 0. 1. 0. 0. 1.]]
A^3 is
[[2. 7. 0. 6. 1. 2. 3.]
 [7. 4. 0. 6. 4. 6. 1.]
 [0. 0. 0. 0. 0. 0. 0.]
 [6. 6. 0. 4. 2. 5. 1.]
 [1. 4. 0. 2. 0. 1. 1.]
 [2. 6. 0. 5. 1. 2. 2.]
 [3. 1. 0. 1. 1. 2. 0.]]
A is not regular
An exception has occurred, use %tb to see the full traceback.

SystemExit
```

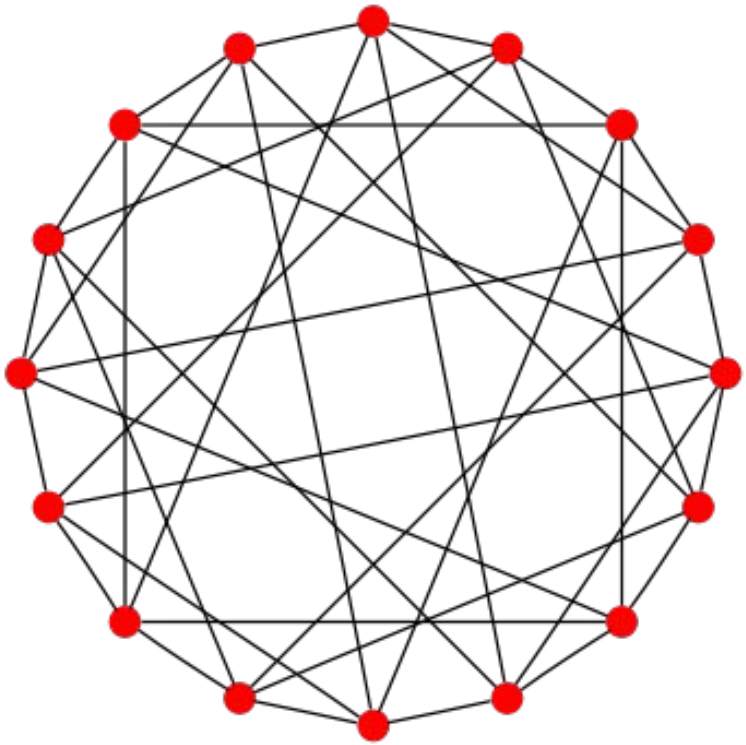
Setting s=4, d=2, C_4 is found from execution:

```
The adjacency matrix A is
[[0. 0. 1. 1.]
 [0. 0. 1. 1.]
 [1. 1. 0. 0.]
 [1. 1. 0. 0.]]
A^2 is
[[2. 2. 0. 0.]
 [2. 2. 0. 0.]
 [0. 0. 2. 2.]
 [0. 0. 2. 2.]]
A^3 is
[[0. 0. 4. 4.]
 [0. 0. 4. 4.]
 [4. 4. 0. 0.]
 [4. 4. 0. 0.]]
A satifies both conditions
```

An attempt to generate all the regular adjacency matrices:

```
[ ]: import numpy as np

def is_valid(matrix):
    # Check if the sum of each row is exactly 3
    row_sums = np.sum(matrix, axis=1)
    return np.all(row_sums == 5)

def backtrack(matrix, i, j, results):
    n = matrix.shape[0]

    # If we've reached the end of the matrix
    if i == n:
        if is_valid(matrix):
            results.append(matrix.copy())
        return

    # Move to the next element
    next_i, next_j = (i, j+1) if j < n-1 else (i+1, i+2)

    # Since the matrix is symmetric, we only need to fill the upper triangle
    if j >= i:
        # Option 1: Set matrix[i][j] and matrix[j][i] to 0
        matrix[i][j] = matrix[j][i] = 0
        backtrack(matrix, next_i, next_j, results)

        # Option 2: Set matrix[i][j] and matrix[j][i] to 1
        matrix[i][j] = matrix[j][i] = 1
        backtrack(matrix, next_i, next_j, results)

        # Reset the element
        matrix[i][j] = matrix[j][i] = 0

def generate_matrices():
    n = 16
    matrix = np.zeros((n, n), dtype=int)
    results = []
    backtrack(matrix, 0, 1, results)
    return results

# Generate all valid matrices
    matrices = generate_matrices()

# Display the number of valid matrices found
    print(f"Number of valid matrices found: {len(matrices)}")

# Example: Display the first valid matrix
    if matrices:
        print("First valid matrix:")
        print(matrices[0])
```

There are no graphs of this type with degree 3 or 4, the next graph of this type is the Clebsch graph, with degree 5 and 16 vertices.



Thank you for listening!