Exploratory Project (3rd semester)

under the guidance of

Prof. S.K. Pandey

Matrix Representation of Graph and Random Walks on the Graphs

Kartik Malik
(20124026)
IDD - 3rd semester

10.12.2021

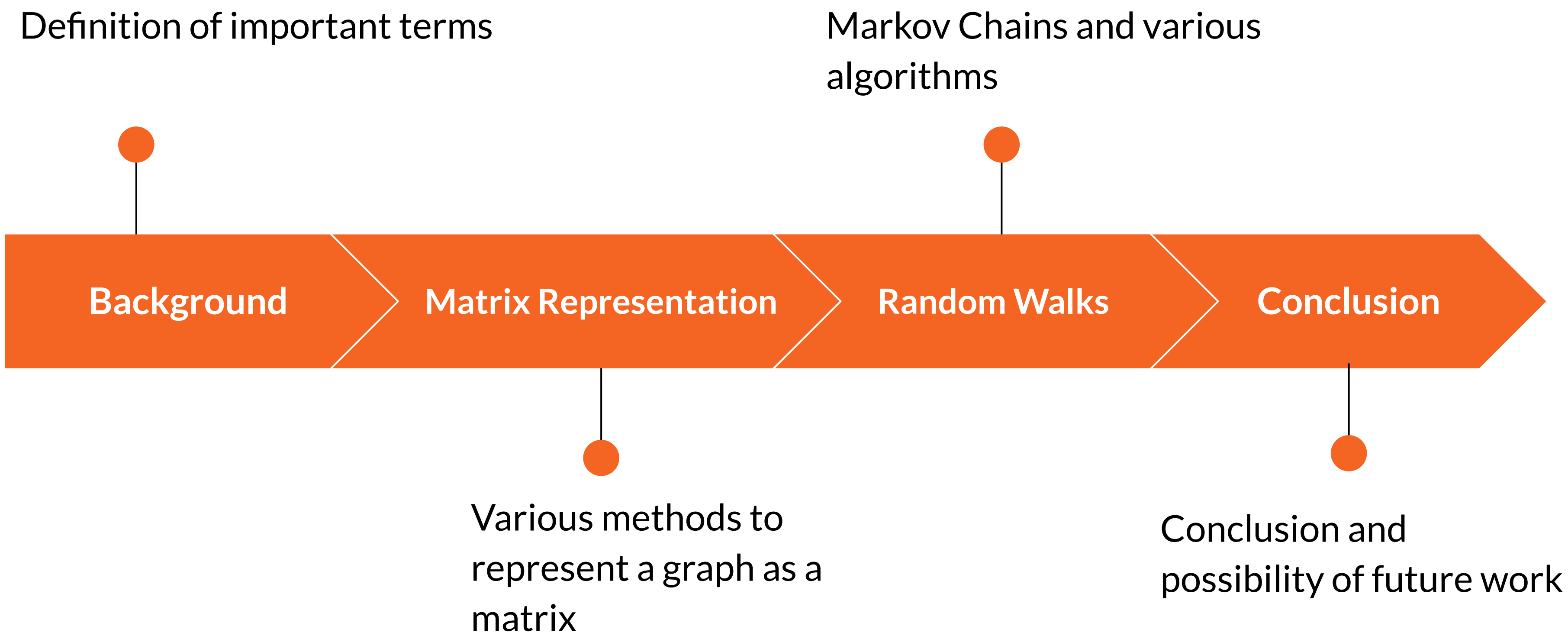
Overview

In this work, we discuss the matrix representation of graph particularly through the adjacency matrix, a simple data structure which can be used for computer processing. We then discuss the random walks on the graphs and see how we can use matrix algebra to easily find the solutions to the problems which would have been tough to answer otherwise.

Motivation

Pictorial representation of a graph, though is a very convenient method for visual study, but other models are better for computer processing. Matrices lend themselves easily to mechanical manipulations and besides, many known results of matrix algebra can be readily applied to study the structural properties of graphs from an algebraic point of view.

Structure of Presentation


Definition of important terms
Markov Chains and various algorithms
Background
Matrix Representation
Random Walks
Conclusion
Various methods to represent a graph as a matrix
Conclusion and possibility of future work

Background Knowledge

Background Knowledge

- **Graph G** : a pair (V, E), where V is a set of vertices, and E is a set of edges between the vertices $E \subseteq \{(u, v) \mid u, v \in V\}$.
- **Undirected Graph** : a graph whose edges are bidirectional, i.e., we can traverse in either direction.
- **Directed Graph** : a graph whose edges have a direction associated with it.
- **Unweighted graph** : a graph in which all the relationships symbolized by the edges are considered equivalent.
- **Weighted graph** : a graph in which each edge has a weight (some real value) which symbolizes the relationship between the nodes, such as the cost, capacity, distance, probability, etc.
- **Simple graph** : an unweighted, undirected graph containing no self-loops or parallel edges.
- **Degree of a vertex** : number of edges that are incident on a vertex, with self-loop counted twice.

Background Knowledge

- **Walk** : a finite alternating sequence of vertices and edges, beginning and ending with vertices, such that each edge is incident with the vertices preceding and following it. No edge appears (or is traversed) more than once in a walk. A vertex, however, may appear more than once.
- **Open walk** : a walk which ends on a different vertex from the one where it starts.
- **Closed walk** : a walk whose first vertex is the same as the last.
- **Rank of a Graph** **r** : the number n − c, where c is the number of connected components of the undirected graph.
- **Rank of a matrix** : number of linearly independent rows or columns in the matrix.
- **Nullity** : the number e - (n − c), in an undirected graph with c connected components and e edges .

Matrix Representation

Incidence Matrix

Definition

It is a two-dimensional matrix in which the rows represent the vertices and columns represent the edges. The entries indicate the incidence relation between the vertex at a row and edge at a column, i.e., in a graph of n vertices and e edges, incidence matrix is a n × e matrix, A = $[a_{ij}]$, where:

$$a_{ij} = \begin{cases} 1, & \text{if } j^{th} \text{ edge } e_j \text{ is incident on } i^{th} \text{ vertex} \\ 0, & \text{otherwise} \end{cases}$$

Properties

- Every edge is incident on precisely two vertices, thus each column of A has exactly two 1′s.
    - Parallel edges will produce identical columns in the matrix.
- The number of 1′s in each row equals the degree of the corresponding vertex.
    - Thus, a row with all 0’s is an isolated vertex.
- Rank of the incidence matrix is the same as the rank of the graph. Thus for a connected graph of n vertices, we need only n − 1 rows of an incidence matrix to specify the corresponding graph completely. Such an $(n − 1) \times e$ submatrix A_f of A is called the *reduced incidence matrix*.
    - Reduced incidence matrix for a tree will be $(n − 1) \times (n - 1)$ matrix, and since its rank is (n - 1), it will be non-singular.

Drawbacks

- It requires space of O(|V|.|E|), which in worst case will result in O($|V|^3$) space complexity. Thus, can only be used to represent graphs with less than 1000 vertices.
- Slow to add or remove vertices and edges, because matrix must be resized/copied. Almost all of the operations, such as, adding edge/vertex or removing edge/vertex require processing of of O(|V|.|E|), which are heavy operations. Thus, the modifications in a graph would be really slow if performed with incidence matrix.

Adjacency Matrix

Adjacency Matrix

Definition

It is a two-dimensional matrix, in which the rows represent source vertices and columns represent destination vertices. Adjacency matrix of an undirected graph G with n vertices and no parallel edges is an n × n symmetric binary matrix $A = [a_{ij}]$ defined such that:

$$a_{ij} = \begin{cases} 1, & \text{if there is an edge between the } i^{th} \text{ and } j^{th} \text{ vertices} \\ 0, & \text{otherwise} \end{cases}$$

Properties

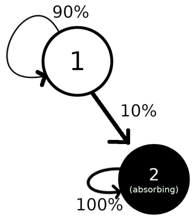
- The adjacency matrix A of a bipartite graph whose two parts have r and s vertices can be written in the form: $A = \begin{pmatrix} 0_{r,r} & B \\ B^{\mathsf{T}} & 0_{s,s} \end{pmatrix},$ where B is an r × s matrix, and $0_{r,r}$ and $0_{s,s}$ represent the r × r and s × s zero matrices. In this case, the smaller matrix B uniquely represents the graph, and the remaining parts of A can be discarded as redundant. B is also known as *biadjacency matrix*.
- Two graphs G_1 and G_2 are isomorphic if and only if there exists a permutation matrix P such that $PA_1P^{-1} = A_2$. In particular, A_1 and A_2 are similar and therefore have the same characteristic polynomial, eigenvalues, determinant, and trace. These can therefore serve as isomorphism invariants of graphs.

Random Walks

Random Walks

- A random walk on a weighted graph is a process that begins at some vertex, and at each time step moves to another vertex randomly with the probability proportional to the weight of the corresponding edge. When the graph is unweighted, the vertex the walk moves to is chosen uniformly at random among the neighbors of the present vertex. Drunkard’s walk is an example of a random walk.
- When the graph is allowed to be directed and weighted, such a walk is called a Markov chain.

Markov Chain



- A Markov chain is a sequence of possible events in which the probability of each event depends only on the present state, i.e., the past affects the future only through the present. Markov chains are often described by a sequence of directed graphs, where the edges of graph are labeled by the probabilities of going from one state at time n to the other states at time n + 1, $P(X_{n+1} = x \mid X_n = x_n)$. The same information is represented by the transition matrix from time n to time n + 1.
  - *Absorbing Markov Chain* is Markov chain in which every state can reach an absorbing state. An absorbing state is a state that, once entered, cannot be left. A state that is not absorbing is called a transient state.

Transition Matrix

- Transition Matrix is a square matrix used to describe transitions of a Markov chain, whose each entry is a non-negative real number representing a probability. Thus, a transition matrix is the same as an adjacency matrix of a directed graph with values in each row summing to 1. If the probability of moving from i to j in one time step is P(j | i) = $P_{i,j,}$ the transition matrix P is:

$$P = \begin{bmatrix} P_{1,1} & P_{1,2} & \dots & P_{1,j} & \dots & P_{1,S} \\ P_{2,1} & P_{2,2} & \dots & P_{2,j} & \dots & P_{2,S} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ P_{i,1} & P_{i,2} & \dots & P_{i,j} & \dots & P_{i,S} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ P_{S,1} & P_{S,2} & \dots & P_{S,j} & \dots & P_{S,S} \end{bmatrix}.$$

Properties

- Since the total of transition probability from a state i to all other states must be 1, i.e. $\sum_{j=1}^{S} P_{i,j} = 1;$ for every i.
- Let X_n has the distribution **Q**, where **Q** is the 1 × S row vector, then $P(X_{n+1}= j) = \sum_i P(X_{n+1}= j \mid X_n = i)$, i.e., $\sum_i Q_iP_{ij}$ which is jth entry of **Q**.P. Thus, the X_{n+1} will have the distribution **Q**.P. Similarly X_{n+2} will have the distribution $Q.P^2$, $X_{n+3} = Q.P^3$, and so on…
    - Similarly, if $P(X_n = i)$, i.e., if we are on a particular state at present, then after m steps, $P(X_{n+m}= j)$ will be the (i, j)th entry of P^m.
- *Canonical form of an absorbing matrix*: Let an absorbing Markov chain with transition matrix P have t transient states and r absorbing states. Then $P = \begin{pmatrix} Q & R \\ \mathbf{0} & I_r \end{pmatrix}$, where Q is a t × t matrix, R is a nonzero t × r matrix, **0** is an r × t zero matrix, and I_r is the r × r identity matrix.

Absorbing Markov Chain

- The expected number of visits to a transient state j starting from a transient state i (before being absorbed) will be entry at (i, j) of inv(I - Q).
  - Proof
    Since the probability of transitioning from i to j in exactly k steps is the entry at (i, j) of Q^k, thus expected number of steps will be (i, j)th entry of $N = I + Q + Q^2 + Q^3 + Q^4 + \dots$, i.e.,
    $$N = 1 / (I - Q) = \mathrm{inv}(I - Q).$$
  - Thus, the expected number of steps before being absorbed when starting at the transient state i will be the sum of N_{ij} for all j, i.e., $N\mathbf{1}$, where $\mathbf{1}$ is a length-t column vector whose entries are all 1.

Random Walks on undirected unweighted networks

The above properties can also be applied to undirected, unweighted graphs since the adjacency matrix of graphs is analogous to the transition matrix of Markov chains.

- If A is the adjacency matrix of the directed or undirected graph G, then the element (i, j) of the matrix A^n gives the number of (directed or undirected) walks of length n from vertex i to vertex j.
  - If n is the smallest non-negative integer, such that for some i, j, the element (i, j) of A^n is positive, then n is the distance between vertex i and vertex j.
- Thus the number of triangles in an undirected graph G is exactly the trace of A^3 divided by 6.
- Matrix Power A^n when calculated by taking the distance product ($c_{ij} = \min(a_{ik} + b_{kj})$, over all k's) gives the shortest walk of n steps from the vertex i to vertex j.

Matrix Exponentiation

- Directly applying the mathematical definition of matrix multiplication gives an algorithm that takes time of the order of n^3 field operations to multiply two $n \times n$ matrices over that field ($O(n^3)$). Thus, A^m would have the complexity $O(n^3m)$.

```
Matrix power(Matrix mat, ll expo) {
      if (expo == 0)
            return Identity(mat.n)[1];
      if (expo == 1)
            return mat;
      Matrix ans = power(mat, (expo - 1));
      ans = mat *[2] ans;
      return ans;
}
```

[1]Identity matrix of size of matrix *mat*

[2]Multiplication of two matrices, code shown in next slide.

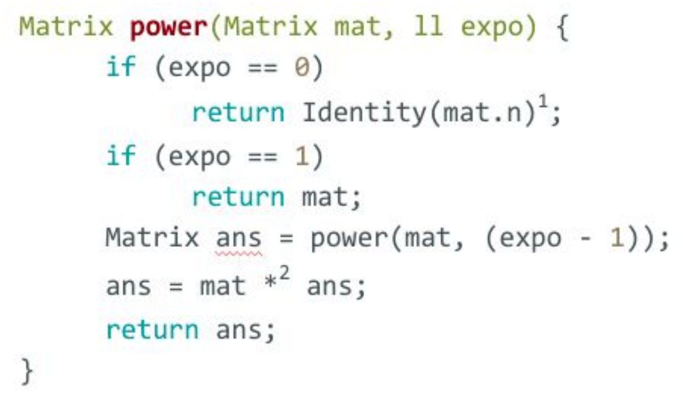
```
struct Matrix {
      vector<vector<int>> result;
      int n;
      Matrix (int _n) : n(_n) {
            result.resize(n);
            for (int row = 0; row < n; row++)
                  result[row].resize(n);
      }

      Matrix operator*(const Matrix& b) const {
            int _n = n;
            Matrix ans(n);
            for (int row = 0; row < n; row++) {
                  for (int col = 0; col < n; col++) {
                        for (int idx = 0; idx < n; idx++) {
                              ans.result[row][col] += result[row][idx] * b.result[idx][col];
                        }
                  }
            }
            return ans;
      }
};
```

Efficient Matrix Exponentiation

We can improve the implementation of exponentiation part, since we don't need to calculate all of A's powers from 1, 2, ..., m. Instead, for calculating A^m, we'll calculate it as $A^{m/2} \times A^{m/2}$ if m is even or $A \times A^{(m-1)/2} \times A^{(m-1)/2}$ if m is odd. Since, at each step we are reducing m at least to half, time complexity improves significantly from linear in m to logarithmic in m, i.e., $O(n^3m)$ to $O(n^3\log m)$.

```
Matrix power(Matrix mat, ll expo) {
      if (expo == 0)
            return Identity(mat.n);
      if (expo == 1)
            return mat;
      Matrix ans = power(mat, (expo >> 1));
      ans = ans * ans;
      if (expo & 1) ans = ans * mat;
      return ans;
}
```



```
Matrix power(Matrix mat, ll expo) {
      if (expo == 0)
            return Identity(mat.n)[1];
      if (expo == 1)
            return mat;
      Matrix ans = power(mat, (expo - 1));
      ans = mat *[2] ans;
      return ans;
}
```

Properties

The number of walks of length n between the vertices i and j is just a_{ij} of A^n. A slight variation to this problem, the shortest walk of n steps between any two vertices can be found by replacing the normal product by the distance product, i.e.,

```
Matrix operator*(const Matrix& b) const {
     int n = _n;
     Matrix ans(n);
     for (int i = 0; i < _n; i++) {
          for (int j = 0; j < _n; j++) {
               ans.val[i][j] = INF; //INF is some large value
               for (int k = 0; k < _n; k++) {
                    ans.val[i][j] = min(ans.val[i][j], val[i][k] + b.val[k][j]);
               }
          }
     }
     return ans;
}
```

Properties

- Another variation of the above problem could be if the walk length is bounded by m instead of being equal to m. If we check the minimum (or doing the other operation, as required) at each power of A from 1 to m, the time complexity will be $O(n^3m)$. However, adding a dummy node and dummy directed edges from all nodes to this node including the self-loop (analogous to absorbing matrix) will solve the problem. Thus making a $(n + 1) \times (n + 1)$ matrix will reduce the time complexity to $O(n^3 \log m)$ in compensation for a slight increase in the space.

Path on chessboard

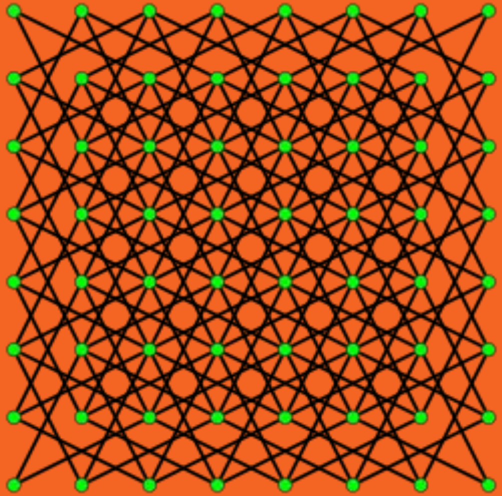


Image source: [Wikipedia]

- Every chess piece has a restricted set of possible moves on the chessboard. Visualising the chessboard as a 64-node graph with the edges representing that the piece has a legal move between the pair of nodes.
    - For example, consider the Knight’s graph which represents all legal moves of the knight chess piece on a chessboard. Each vertex of a knight’s graph represents a square on a chessboard, and each edge represents a legal move from one square to another.

Data Structure

- The adjacency matrix may be used as a data structure for the representation of graphs in computer programs for manipulating graphs. Because each entry in the adjacency matrix requires only one bit, it can be represented in a very compact way, occupying only $|V|^2/8$ bytes to represent a directed graph, or (by using a packed triangular format and only storing the lower triangular part of the matrix) approximately $|V|^2/16$ bytes to represent an undirected graph.

Adjacency List

Definition

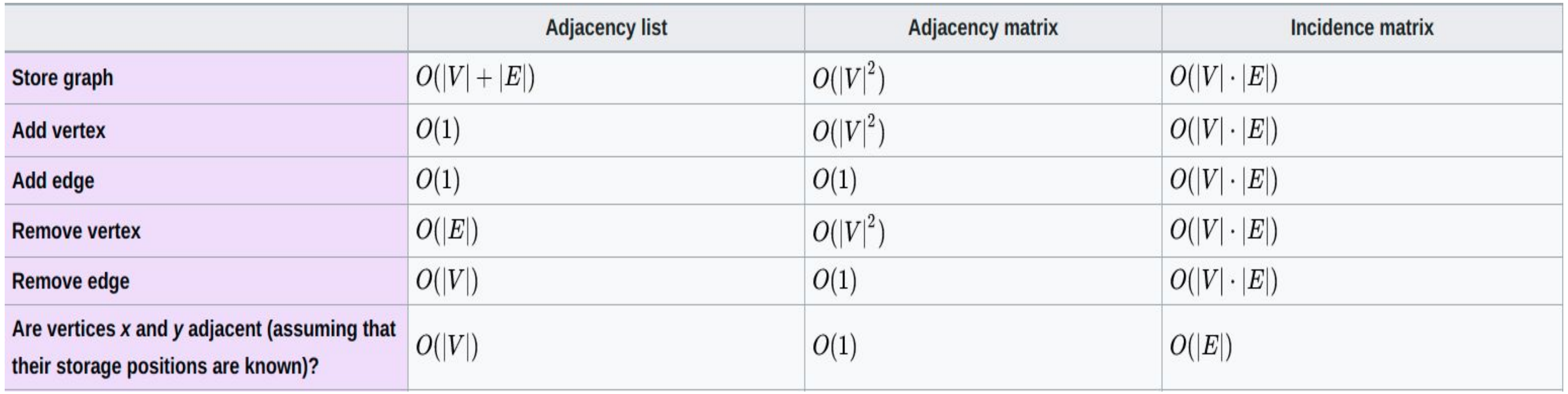
An adjacency list is a collection of unordered lists used to represent a finite graph. Each unordered list within an adjacency list describes the set of neighbors of a particular vertex in the graph.

It’s the main alternative of adjacency matrix for representing a graph, and is quite efficient when dealing with sparse graphs compared to adjacency matrix.

Results

Experimental Results

- A matrix is an optimal way to represent a graph to a computer for computer processing.
- Many known results of matrix algebra can be readily applied to study the structural properties of graphs from an algebraic point of view.
- Comparison between common data structures for performing common operations



	Adjacency list	Adjacency matrix	Incidence matrix
Store graph	$O(\|V\|+\|E\|)$	$O(\|V\|^2)$	$O(\|V\|\cdot\|E\|)$
Add vertex	$O(1)$	$O(\|V\|^2)$	$O(\|V\|\cdot\|E\|)$
Add edge	$O(1)$	$O(1)$	$O(\|V\|\cdot\|E\|)$
Remove vertex	$O(\|E\|)$	$O(\|V\|^2)$	$O(\|V\|\cdot\|E\|)$
Remove edge	$O(\|V\|)$	$O(1)$	$O(\|V\|\cdot\|E\|)$
Are vertices *x* and *y* adjacent (assuming that their storage positions are known)?	$O(\|V\|)$	$O(1)$	$O(\|E\|)$

Image Source: Wikipedia

Conclusion

1. The adjacency matrix is more efficient method for supporting the modifications to the graph compared to the incidence matrix.
2. Adjacency list is the better alternative when dealing with sparse graphs than the matrix.
3. We can use matrix properties for solving many problems, which would have been complicated otherwise.

Scope for Future Work

Graphs with trillions of edges occur in many modern fields such as, machine learning, social network analysis, and in many other areas. Efficient compressed graph representations need to be developed to reduce the I/O and memory requirements.

Thank You