Pruning Techniques for Graph Isomorphism: Monte Carlo, Lanczos and BFS

Zi-kai Lin, Ruo-zhou Du

January 9, 2025

Abstract

In this paper, we explored the pruning techniques for graph isomorphism (GI) from three ways: Monte Carlo approach based on Perron-Frobenius theorem, Lanczos pruning based on Cauchy interlacing theorem and BFS pruning based on BFS with depth. We find Monte Carlo approach was sensitive to noise and not suitable for GI while Lanczos and BFS algorithm showed great pruning ability in our experiments. We suggested that future research related to GI may combine methods from different fields including statistical computation, matrix algebra and classical discrete mathematics.

Contents

1 Introduction

An isomorphism between two graphs is a *bijection* between their vertex sets that *preserves adjacency*. The *graph isomorphism problem (GI)* is that of determining whether there is an isomorphism between two given graphs[1]. GI focuses on the structural equivalence between two graphs, while vertex labels and edge labels may not be consistent.

Computing isomorphism for two given graphs G and H is a combinatorial search task. In the worst case, it amounts to searching all $|V|!$ combinations of nodes $v \in V$. While the problem of subgraph isomorphism is known to be NP-complete, this question is still open for GI[2], i.e., no polynomial time algorithm is known for GI. To our knowledge, the fastest proven running time for GI has stood for three decades at $e^{\mathcal{O}\sqrt{n \log n}}$ [3].

Unfortunately, the high complexity results in problems related to graph retrieval in the case of large-scale graph database query. When a certain query graph q comes, if we run the GI algorithm between query graph q and all data graphs $\{g_i\}_{i=1}^N$ in database, the exponential complexity will be unbearable. Therefore it is significant to develop some pruning techniques to reduce the number of candidates in database before standard GI algorithm is called, e.g. VF2 implemented in Python package `networkx`.

In this paper, we assume the graphs $\{g_i\}_{i=1}^N$ in database are relatively static, whose features can ben extracted offline before query. On the other hand, the query graph q is dynamic and streaming with unknown distribution. Our pruning techniques will make use of features of $\{g_i\}_{i=1}^N$ extracted in advance to reduce the number of candidates.

Since most graphs emerged from reality are sparse, we focus on graphs with *linear complexity*, i.e., $|E| = \mathcal{O}(|V|)$. In our experiments, when preparing random graphs for test, whether to generate a edge between vertice i and j was a Bernoulli random variable with probability $p = \mathcal{O}(\frac{1}{|V|})$, which guaranteed the linear complexity of graphs. For example, setting $p = \frac{6}{|V|}$ means each vertex has 6 neighbours in average.

In addition, we suppose all the graphs of our interest are *weakly connected*. A weakly connected graph is a directed graph in which it is possible to reach any node starting from any other node by traversing edges in some direction (i.e., not necessarily in the direction they point). Weakly connected graphs are common in the field of organic chemistry and social network. We claim this assumption does not decrease the utility of our pruning techniques. If a query graph q is not weakly connected, first use standard *strongly connected component* (SCC) detection algorithm, e.g., Tarjan, to take it apart, and use the number and sizes of SCC as a feature to prune. Only the graphs in database with the same number and sizes of SCC will be taken into consideration. And then run our pruning algorithm on each pair of SCC of the same size between q and graphs in database. Thus it is reasonable to assume all the graphs of our interest is weakly connected.

The outline of the rest of this paper is as follows. The next section looks at pruning by Monte Carlo. Some related matrix and graph theory are reviewed in order to develop a particle simulating method that estimates the Perron vector corresponding to Perron root for pruning. The subsequent section shows the motivation of using Lanczos algorithm to prune due to the sensitivity and failure of Monte Carlo. We reveals the important role of Cauchy interlacing theorem in Lanczos pruning and the pros and cons of Lanczos pruning are also discussed. A fifth section looks at another method to prune based on BFS. The method integrates static structural features (degree distribution) with dynamic positional information (depth sequences obtained through BFS from certain nodes) to characterize graph topology, enabling discrimination between non-isomorphic graphs that share identical major features. The final section summarizes and suggests some avenues for future research.

2 Monte Carlo

In this section, we first transform the graph into a irreducible Markov chain. By Perron-Frobenius theorem, we proposed that the unique stationary states of the irreducible Markov chain can be utilized as a pruning feature. Futhermore we suggested that using particle simulation to esimate the stationary states, i.e., Monte Carlo method, might be a potential approach with relatively low complexity. Subsequently some experiments were conducted and we found Monte Carlo approach was not suitable for GI problem.

2.1 Connection between graph and Markov chain

Here we claim some notations that will be used later in this paper. A is the weighted adjacency matrix with all elements non-negative. If $A_{ij} > 0$, then there exists an directed edge from vertex i to j. M is the indicator matrix, or the traditional adjacency matrix, where all elements belong to $\{0, 1\}$. $M_{ij} = 1$ if $A_{ij} > 0$ else $M_{ij} = 0$. For an undirected graph, the corresponding indicator

matrix M is symmetric. If we normalize each row of A to sum up to 1 (or normalize each row of M), we obtain the transition matrix P of a Markov chain, where $P_{ij} = \mathbb{P}(x_{t+1} = s_j | x_t = s_i)$ and one step of transition can be denoted as $x_{t+1} = P^\top x_t$.

If we put some particles into the graph and let them flow according to the Markov chain, the particles can explore the graph and collect information[1]. We can expect that the behaviour of these particles may reflect some features for pruning.

Notice that the graphs of our interest is weakly connected. This means if we turn it into an *undirected* graph, then the corresponding matrix M and A will both be *irreducible*, i.e., there exists no permutation matrix Q such that $QMQ^\top$ or $QAQ^\top$ is a block upper triangular matrix. By Perron-Frobenius theorem, the normalized transition matrix P will have some attractive features:

1. The spectrum radius of P is 1, called Perron root.
2. There exists a unique positive eigenvector corresponding to Perron root, called Perron vector[4].

which guaranteed that the stationary distribution of particles in an irreducible Markov chain will also be unique. Therefore we consider strongly connected undirected graphs in this section. If the given graph is weakly connected, it is very easy to turn it into a strongly connected graph: for any edge from vertex i to vertex j, just add its reverse edge (from vertex j to vertex i) into the given graph.

2.2 Setup for Monte Carlo approach

Denote $x_0 \in \mathbb{N}^{|V|}$ as the initial distribution of particles, and x_k as the distribution of particles after k steps of transition. All the elements in x_k are non-negative integars, representing the number of particles in a certain vertex after k steps. One step of transition is denoted as $x_{t+1} = T(x_t)$. Actually, the transition of particles in one step is an approximation of $x_k \approx P^\top x_t$, with all the elements truncated into integars.

As for a certain particle in vertex i, during one step of transition, it randomly chooses a neighbour of vertex i, say, j_1, with the probability of

$$P_{ij_1} = \frac{1}{Z} d_{j_1}^2$$

where d_{j_1} is the degree of vertex d_{j_1} and Z is a normalizing constant:

$$Z = \sum_{j_k \in \mathcal{N}(i)} d_{j_k}^2$$

where $\mathcal{N}(i)$ denotes all the neighbours of vertex i. The motivation behind the transition probability is that vertex with higher degree contains more structual information and is worth exploring more frequently.

Here is an example. Suppose in Fig1, one particle is currently in vertex i whose neighbours are $\{j_1, j_2, j_3\}$. The degrees are $\{5, 3, 1\}$ and the square of degree are $\{25, 9, 1\}$ respectively. Then the transition probability of this particle from current vertex i to next vertex is

$$P_{ij_1} = \frac{25}{35},\ P_{ij_2} = \frac{9}{35},\ P_{ij_3} = \frac{1}{35}.$$

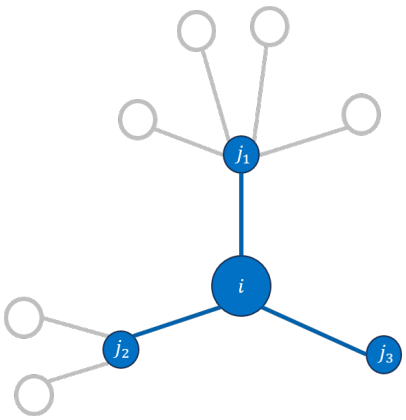


Figure 1: Transition Example

2.3 Experiments

Metric for convergence When the distribution of particles converges, we have $x_{t+1} \approx x_t$. To make the distribution free of the total number of particles, we define

$$v_{t+1} = \frac{x_{t+1}}{\|x_{t+1}\|_1},\ v_t = \frac{x_t}{\|x_t\|_1}$$

where the denominators is the 1-norm representing the total number of particles and remains unchanged during transition.

[1] We appreciate Yi Li for coming up with this interesting idea.

After normalinzing, v_t can be considered as a probability mass function (pmf).

Now we define the update of distribution in the t-th step as

$$e_t \triangleq \text{dist}(v_t, v_{t-1}).$$

There exist many kinds of distance function as measurement for two normalized vectors. We focus on the following three kinds:

1. 2-norm;
2. JS divergence;
3. Wasserstein distance;

The latter two kinds of distance are usually used to estimate the distance between two distributions in probability theory.

In the first experiment, we set the graph size $|V| = 200$, the probability to randomly generate edges $p = \frac{6}{|V|}$ and the total number of particles $N = 20 \times 200 = 4000$. We conducted 50 steps of transition (long after convergence). We compared the updates $\{e_i\}_{i=1}^{50}$ under the three metrics and results are shown in Fig2. To make them comparable, we normalized them by setting $e_1 = 1$ for the three metrics.

In Fig2, all of the three metrics indicated that the distribution of particles converged within 10 steps. However, the updates $\{e_i\}_{i=1}^{50}$ measured by 2-norm and Wasserstein distance (blue and green line) fluctuated more dramatically than the updates measured by JS divergence (orange line). In practice, we need to find a *convergence threshold* D measured by some distance such that we stop iteration as long as the update e_k is less than D. Hence JS divergence is prefered since it has smaller variation after the transition converges and can indicate convergence more precisely than 2-norm and Wasserstein distance. In our following experiments, we choose JS divergence as the dist$(\cdot, \cdot)$ function.

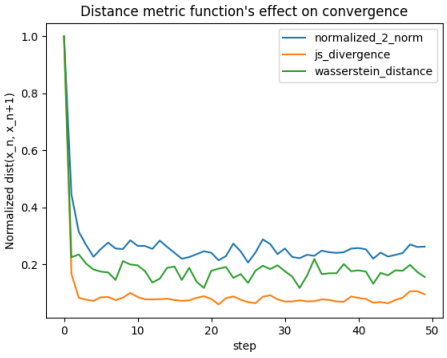


Figure 2: Three metrics of updates

Convergence threshold In this experiment, we explored how the convergence threshold varied as $|V|$ increases. Given $|V|$ and p, the convergence threshold is defined as

$$D = \mathbb{E}_g \left[\lim_{t\to\infty} \mathbb{E}[\text{dist}(v_t, v_{t-1})|g] \right] = \mathbb{E}_g \left[\lim_{t\to\infty} \mathbb{E}[e_t|g] \right].$$

where g represents strongly connected undirected graphs of $|V|$ vertice and of randomly generated edges with probability p.

In practice, we randomly generated many graphs $\{g_i\}_{i=1}^{M}$. And for a given graph g_i, let T be large enough, say $T = 50$ and use the average of the last 5 updates:

$$\hat{e}^{(g_i)} = \frac{1}{5} \sum_{t=T-4}^{T} e_t^{(g_i)}$$

as an approximation of $\lim_{t\to\infty} \mathbb{E}[e_t|g]$. And use the mean of all the generated graphs:

$$\hat{D} = \sum_i \hat{e}^{(g_i)}$$

as the estimator of D.

We tried $|V| = 25, 50, \ldots, 300$ and set $p = \frac{6}{|V|}$, the total number of particles $N = 20 \times |V|$ and the number of randomly generated graphs $M = 20$. The results are shown in Fig3

In Fig3, we can observe that the convergence threshold D seems uncorrelated with $|V|$. Another phenomenon worth mentioning is that the scale of convergence threshold D is around 0.012. Recall that D can be seen as the average of truncation error arising from the Monte Carlo approximation of matrix-vector multiplication $x_{t+1} = P^\top x_t$. Therefore the scale of D can also been considered as a kind of noise in simulation and it is reasonable that the scale of noise is independent of $|V|$.

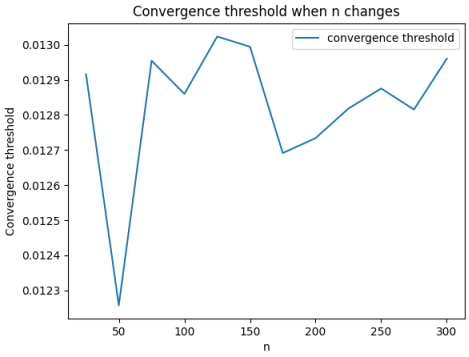


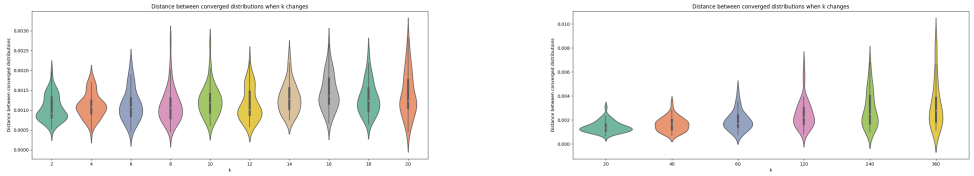
Figure 3: Convergence threshold against $|V|$

Difference between stationary distributions of two distinct graphs In this experiments, we looks at the pruning ability of stationary states.

Suppose we have two distinct graph G_1 and G_2, and their positive eigenvectors corresponding to 1 are denoted as $c_1 x_1$ and $c_2 x_2$, where c_1, c_2 are constant and $G_1 x_1 = x_1, G_2 x_2 = x_2$.

Suppose the normalized stationary particle distribution of the Markov chain induced by G_1 and G_2 are u_1 and u_2, with $\|u_1\|_1 = \|u_2\|_1 = 1$. Since u_1 and u_2 are approximation of x_1 and x_2, we can expect that $\text{dist}(u_1, u_2)$ may reflect the difference of the two distinct graphs.

We still set $|V| = 200, p = \frac{6}{|V|}$ and the total number of particles $N = 20 \times |V|$. For a given graph G_1, we randomly perturbed k pair of edges to get another graph G_2 (Since we consider undirected graph in this section, then the edge from i to j and the reverse edge from j back to i should be perturbed simultaneously). Then Monte Carlo simulation were conducted on G_1 and G_2. For given $|V|$, the simulation were randomly repeated 50 times. The mean of the difference between normalized stationary particle distribution $\text{dist}(u_1, u_2)$ against $|V|$ are shown in the form of violinplot in Fig4.



(a) small k

(b) large k

Figure 4: Difference between stationary particle distribution against $|V|$

We tried perturbation with small $k \in [2, 20]$ and the results are shown in Fig4(a). The difference $\text{dist}(u_1, u_2)$ seemed to slightly increases as k increases. We also tried large $k = 40, 60, 120, 240, 360$ and the results are shown in Fig4(b). The tendency is similar and outliers are more likely to appear when k is larger.

2.4 Discussion of Monte Carlo approach

First, the pruning ability of Monte Carlo method is not strong. The scale of $\text{dist}(u_1, u_2)$ is worth mentioning. Even if k as large as 360 ($\frac{k}{|E|} \approx 30\%$), the mean of $\text{dist}(u_1, u_2)$ is around 0.004. Notice that $\text{dist}(u_1, u_2)$ can be seen as the signal indicating the difference of two graphs, then the signal-noise rate (SNR) is around $\frac{0.003}{0.012} = 25\%$. This result implies that the pruning ability of stationary particle distribution is not enough and the low SNR is expected to yield a high false positive rate, i.e., two isomorphic graphs may be mistaken as not isomorphic due to the unavoidable update e_i (noise) in simulation.

Second, Monte Carlo approach does not save the computation cost. The motivation behind Monte Carlo approach is to use the Perron vector corresponding Perron root (in this case, Perron root is 1) as the pruning feature. Since the Perron vector is unique, why not use power method[5]? The power method $x_k = (P^\top)^k x_0$ can quickly obtain a good approximation of the Perron vector. Since the graph is sparse and $|E| = \mathcal{O}(|V|)$, then the complexity of one step of matrix-vector multiplication $x_k = P^\top x_{k-1}$ is $\mathcal{O}(|V|)$. On the other hand, in one step of transition, the complexity is proportional to the number of particles N. In practice, we usually set $N = \mathcal{O}(|V|)$ to avoid dramatic fluctuation during transition. Then the complexity in one step of transition is also $\mathcal{O}(|V|)$. Hence Monte Carlo, compared with power method, does not save the computation cost.

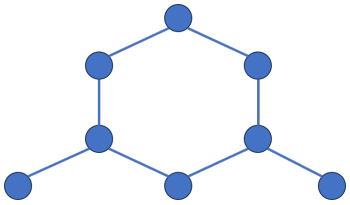


Figure 5: An example of how Monte Carlo approach failed

Third, since the adjacency matrix A and the induced transition matrix P are just irreducible non-negative matrices, Perron-Frobenius *cannot* guarantee that $\forall \lambda \neq 1, |\lambda| < 1$. This leads to a severe practical problem that the power method or the Monte Carlo simulation may not converge.

For example, for the symmetric graph in Fig5, the corresponding transition matrix P has symmetric eigenvalue $\pm\lambda_1, \pm\lambda_2, \ldots, \pm\lambda_n$. Perron vector, or the stationary practicle distribution of P is still unique. However, since the eigenvalue is symmetric, -1 is also the eigenvalue of P. Then the eigenvector corresponding to -1 is the Perron vector of P^2:

$$P^\top x_2 = -x_2,\ (P^\top)^2 x_2 = x_2$$

which fails the power method. Even though Monte Carlo approach restricts the pariticle distribution to be non-negative, we still observed the following phenomenon in particle simulation:

$$u = T(v),\ v = T(u)$$

we called this phenomenon *periodic dangling*, which failed Monte Carlo approach.

3 Lanczos pruning

Monte Carlo approach only uses the largest eigenvalue and the corresponding eigenvector to prune. Can we use more eigenvalues and eigenvectors to prune? Motivated by this idea, in this section, we developed another kind of pruning algorithm based on Cauchy interlacing theorem and Lanczos iteration, which take adavantage of the sparsity of graph and is proved to have strong pruning ability. We will first review some matrix theory and numerical linear algebra. Subsequently, the details of Lanczos pruning algorithm and experiments are followed. Results of experiments validated the great pruning power of our Lanczos pruning algorithm. And finally pros and cons are discussed.

3.1 Revist interlacing theorem and Lanczos iteration

We first reviewed the Cauchy interlacing theorem[4], which provides the theoretical fundation for Lanczos pruning.

Theorem 1 (Cauchy Interlacing Theorem) *Let $A \in \mathbb{C}^{n\times n}$ be Hermitian matrix. Suppose that $1 \leq m \leq n$, and $U \in \mathbb{C}^{n\times m}$ be orthonormal, i.e., $U^*U = I_m$. Denote the eigenvalues of A as $\lambda_1, \lambda_2, \ldots, \lambda_n$, eigenvalues of U^*AU as $\mu_1, \mu_2, \ldots, \mu_m$, which are arranged in ascending order. Then*

$$\lambda_i \leq \mu_i \leq \lambda_{i+n-m}, \quad , i = 1, \ldots, m$$

Intuitionally, Cauchy interlacing theorem says that squeezing a Hermitian matrix by orthonormal matrix will also squeeze the eigenvalues. Therefore, if we can squeeze the adjacency matrix A, which are assumed to be symmetric (Hermitian) because we only consider undirected strongly connected graph, then the eigenvalues of the squeezed matrix, i.e., $\{\mu_i\}_{i=1}^m$, can be used to prune.

Now we reviewed the framework of Lanczos iteration[5]. Lanczos iteration is a classical iterative algorithm for sparse Hermitian matrix. The basic idea is:

$$AQ_n = Q_{n+1}\tilde{T}_n, \quad T_n = Q_n^* A Q_n$$

where Q_n and Q_{n+1} are orthonormal matrix and $Q_{n+1} = [Q_n, q_{n+1}]$. T_n is a symmetric tridiagonal matrix:

$$T_n = \begin{bmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \beta_2 & \alpha_2 & \ddots & \\ & & \ddots & \ddots & \beta_{n-1} \\ & & & \beta_{n-1} & \alpha_n \end{bmatrix}$$

Notice that the eigenvalues of symmetric tridiagonal matrix is very easy to calculate.

Details of Lanczos iteration are shown in Algorithm 1, where A is assumed real and symmetric.

Algorithm 1 Lanczos iteration

Require: $A^\top = A$
$\beta_0 \leftarrow 0$
$q_0 \leftarrow 0$
$b \leftarrow$ arbitrary nonzero vector
$q_1 \leftarrow b/\|b\|_2$
for $n = 1, 2, 3, \dots$ **do**
  $v \leftarrow Aq_n$
  $\alpha_n \leftarrow q_n^\top v$
  $v \leftarrow v - \beta_{n-1}q_{n-1} - \alpha_n q_n$
  $\beta_n \leftarrow \|v\|_2$
  $q_{n+1} \leftarrow v/\beta_n$
end for

Notice that in practice, there is no need to store Q_n. Only q_{n-1} and q_n should be stored. And we do not need to explicitly store matrix T_n since it is symmetric and sparse. We only need to store two vectors that contains $\{\alpha_1, \alpha_2, \dots\}$ and $\{\beta_1, \beta_2, \dots\}$.

3.2 Setup for Lanczos pruning

Given two distinct graph G_1 and G_2, their weighted adjacency matrices are denoted as A_1 and A_2, their indicator matrices are denoted as M_1 and M_2. Here we choose the weight $A_{ij} = \frac{1}{2}(d_i + d_j)$, where d_i and d_j are the degree of vertex i and j respectively.

Suppose that G_1 is in the graph database and the eigenvalues of A_1 and M_1 can be calculated offline in advance while G_2 is a query graph. Given G_2, we can carry out Lanczos iteration on A_2 (or M_2) and calculate the eigenvalues of $T_1, T_2, \dots$. Since the interlacing property of T_n is a necessary condition for isomorphism between G_1 and G_2, we can stop iteration once we observe the interlacing property is violated and prune G_1 out of candidates.

However, Lanczos iteration has a drawback: it is numerically unstable. When n increases, the columns of Q_n will gradually lose orthonormal and the interlacing property will be violated even if G_1 and G_2 are isomorphic. This drawback may lead to false positive result if pruning cannot be done before Lanczos iteration loses orthonormal. We will look at the problem in the following experiments.

3.3 Experiments

Iters when Lanczos iteration loses orthogonality In this experiment, we looked at the mean iteration when Lanczos iteration loses orthogonality. We also compared how weighted adjacency matrix A and indicator matrix M can prevent Lanczos iteration from losing orthogonality.

We tried $|V| = 100, 200, \dots, 1200$ and set $p = \frac{6}{|V|}$. For certain $|V|$, we randomly generated 25 graphs with corresponding A and M, and perform Lanczos iteration on A and M. We calculated the average of steps when Lanczos iteration loses orthogonality. The results are shown in Fig6(a). The red line corresponds to weighted adjacency matrix A and the blue line corresponds to indicator matrix M.

We can see that the mean of steps when Lanczos iteration loses orthogonality is increasing as $|V|$ increases. When $|V|$ is small, say $|V| \leq 300$, indicator matrix M (blue line) can prevent loss

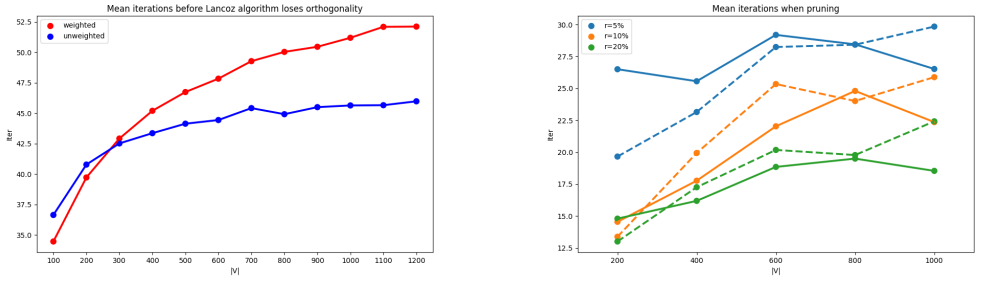
of orthogonality better whereaas when $|V|$ is large, weighted adjacency matrix A (red line) can better prevent loss of orthogonality.

Iters when Lanczos pruning stops In this experiment, we looked at the difference of pruning ability between A and M as the the proportion of perturbation varied. We tried $|V| = 200, 400, \ldots, 1000$, and still set $p = \frac{6}{|V|}$. Given an original graph G_1, we randomly perturbed $r \times |E|$ edges to make a distinct graph G_2. We tried different perturbation rate $r = 5\%, 10\%, 20\%$.

Given $|V|$ and r, we randomly generated 10 original graphs. For a certain original graph, we randomly generated 10 perturbed graphs for test. We calculated the mean of steps when Lanczos iteration finished pruning. The results are shown in Fig6(b). The dotted line corresponds to indicator matrix M while the solid line corresponds to weighted adjacency matrix A.

First, we observed that the average of steps when Lanczos iteration finished pruning is creasing as $|V|$ increases and r decreases. This is reasonable since larger $|V|$ and smaller r yield harder GI problem and Lanczos iteration needs more steps to distinguish two graphs.

Second, we found that when r is small, Lanczos iteration performed on M needs less steps to prune. Nonetheless, when r is large, Lanczos iteration performed on A needs less steps to prune.



(a) Iters when Lanczos loses orthogonality

(b) Iters when Lanczos iteration finish pruning

Figure 6: Iters of Lanczos pruning

3.4 Discussion of Lanczos pruning

The adavantages of Lanczos pruning includes:

1. High efficiency. Lanczos pruning often detect distinction when T_n is still small even if $|V|$ is as large as 1000.
2. Robustness. Lanczos pruning still works even when the perturbating proportion r is little.
3. Low time complexity. Lanczos iteration makes good use of sparsity. In Algorithm1, in each step of iteration, the most complex step is $v \leftarrow Aq_n$, with time complexity $\mathcal{O}(|V|)$. Moreover, it is very easy to estimate the eigenvalues of tridiagonal matrix T_n. Notice that T_n is often small matrix when finish pruning, the cost to calculate the eigenvalues of $T_1, T_2, \ldots, T_n$ is also small.
4. Low space complexity. In practice, we only need to store q_{n-1}, q_n, $\{\alpha_1, \alpha_2, \ldots\}$ and $\{\beta_1, \beta_2, \ldots\}$, with space complexity $\mathcal{O}(|V|)$.

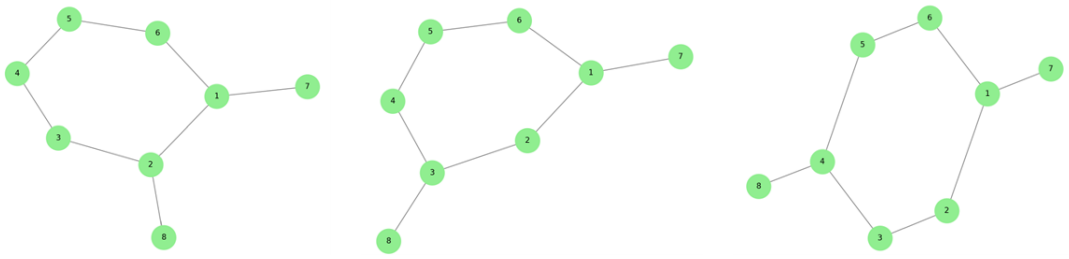
The disadvantage of Lanczos pruning is that Lanczos iteration usually loses orthogonality when n is large, which can lead to false positive result. However, compare Fig6(a) and Fig6(b), Lanczos iteration always finishes pruning long before it loses orthogonality. Therefore, we can conclude that Lanczos pruning is unlikely to yield false positive result and can prune safely.

4 BFS pruning

4.1 Inspiration

When studying Monte Carlo algorithms for graph isomorphism problems, I often consider how to use existing data to describe graph characteristics and enhance the algorithm's ability to identify

graphs. For a graph, the number of nodes, edges, and degrees of each node are readily available features, which is why I incorporated degrees into the Monte Carlo weight settings. However, these features can only describe the overall structure of the graph and cannot capture the internal positional relationships. For example, in chemistry, the problem of structural isomers can only be distinguished through internal positional relationships. Therefore, I came up with the idea of using Breadth-First Search (BFS) to label the depth of each node to describe the positional information within the graph.



(a) (b) (c)

Figure 7: three isomers of dimethylbenzene (xylene)

In the subsequent discussion, we only consider cases where the number of nodes, edges, and degrees are identical, because if these conditions are not met, it is easy to conclude non-isomorphism. Additionally, we do not consider the labels of nodes themselves, focusing instead on structural isomorphism. We use the three isomers of dimethylbenzene (xylene) in Fig7 as an example. They share the same information that:

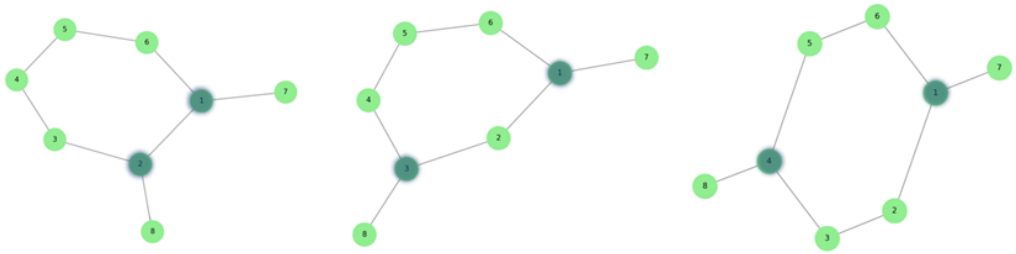
- Node: 8
- Edge: 8
- Degree: 1, 1, 2, 2, 2, 2, 3, 3

The current objective is to perform BFS on these graphs to obtain depth information for all nodes relative to certain source node(s). The selection of these source nodes must satisfy the following condition: if two graphs are isomorphic, then the source nodes from both graphs must be equivalent, ensuring that the BFS results are identical under isomorphism. If two graphs are non-isomorphic, we desire that the BFS results from the source nodes should be as different as possible to facilitate the verification of non-isomorphism.

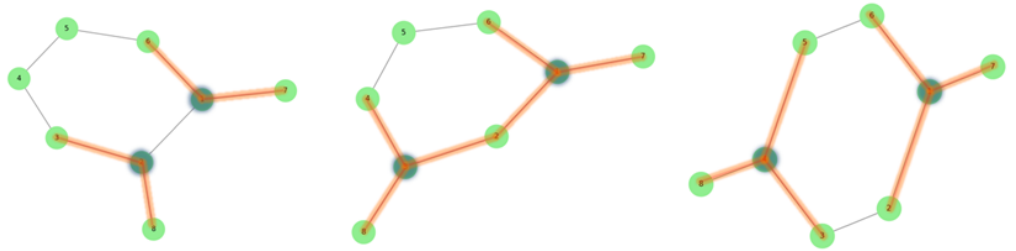
Based on these conditions, I propose a typical approach for selecting source nodes: choose all nodes with a specific degree value, using the same degree value for both graphs. This ensures that when two graphs are isomorphic, the sets of source nodes selected from each graph are equivalent.

The choice of this specific degree value is arbitrary and will be discussed in detail later. However, selecting a sufficiently distinctive degree value can make the BFS results more distinguishable. Therefore, I prefer to choose the maximum degree in the graph, as statistically, the number of nodes with the maximum degree tends to be relatively small.

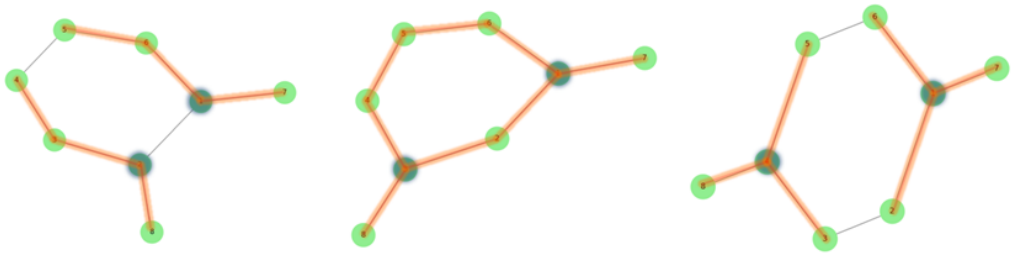
Taking the simplified graphs of the three isomers of dimethylbenzene as examples, we performed BFS starting from the nodes with maximum degree (d=3) in Fig8:



(a) First step



(b) Second step



(c) Third step

Figure 8: BFS performed on three isomers of dimethylbenzene

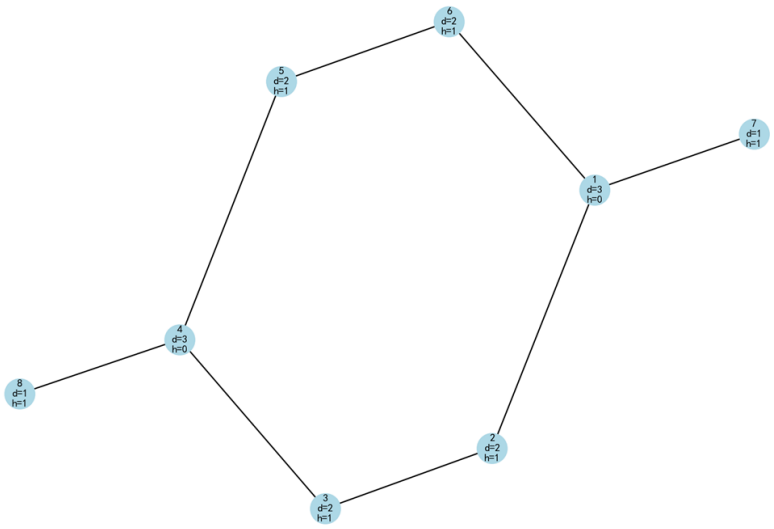


Figure 9: Degree-depth combinations as new features

After three steps, we can obtain the depths of all nodes relative to the nodes with maximum degree. To enhance distinguishability, since the degrees of all nodes are known, we combine degree and depth information, using degree-depth combinations as new features for each node.

Take the third graph as an example (shown in Fig9). By listing all degree-depth combinations

in the graph, we obtain the final results, which can serve as a basis for determining graph non-isomorphism (regardless of order). The results for the three graphs are as follows:

- `[1,1], [1,1], [2,1], [2,1], [2,2], [2,2], [3,0], [3,0]`
- `[1,1], [1,1], [2,1], [2,1], [2,1], [2,2], [3,0], [3,0]`
- `[1,1], [1,1], [2,1], [2,1], [2,1], [2,1], [3,0], [3,0]`

We can observe that although the three graphs maintain consistency in their three major characteristics, they differ in the frequency of `[2,1]` and `[2,2]` combinations in this algorithm, thus leading to the conclusion that they are non-isomorphic.

4.2 Algorithm

Below is the complete pseudocode for this algorithm. [2]

Algorithm 2 DegreesDepthCombination

Require: Graph G, initial degree x
Ensure: List of degree-depth combinations

for each node v in G.nodes **do** ▷ Calculate degrees for all nodes
    degree[v] ← calculate_degree(v)
end for
N ← empty list ▷ Find initial nodes with degree x
for each node v in G.nodes **do**
    if degree[v] $= x$ **then**
        add v to N
    end if
end for
R ← empty dictionary ▷ Initialize result dictionary R
for each node v in G.nodes **do**
    $R[v] \leftarrow [\text{degree}[v], \infty]$
end for
Queue ← empty queue ▷ Multi-source BFS from nodes in N
for each node n in N **do**
    $R[n][1] \leftarrow 0$ ▷ Set initial depth
    enqueue n to Queue
end for
while Queue is not empty **do**
    current ← dequeue from Queue
    current_depth ← R[current][1]
    for each neighbor v of current **do**
        if $R[v][1] = \infty$ **then**
            $R[v][1] \leftarrow$ current_depth $+ 1$
            enqueue v to Queue
        end if
    end for
end while
Result ← list(R.values) ▷ Convert dictionary to list of values
return Result

[2] We sincerely apologized for the rough format of the rest of this paper due to limited time.

Algorithm 3 CompareGraphStructures

Require: Graphs G_1, G_2, optional initial degrees $x_1 = \text{null}$, $x_2 = \text{null}$
Ensure: Boolean indicating if structures are equivalent

if $x_1 = \text{null}$ **then**
    $x_1 \leftarrow \text{max\_degree}(G_1)$ ▷ If x_1 not specified, use maximum degree in G_1
end if
if $x_2 = \text{null}$ **then**
    $x_2 \leftarrow \text{max\_degree}(G_2)$ ▷ If x_2 not specified, use maximum degree in G_2
end if
▷ Actually $x_1 = x_2$ because of the conditions above
$Result_1 \leftarrow \text{DegreesDepthCombination}(G_1, x_1)$ ▷ Get degree-depth combinations for G_1
$Result_2 \leftarrow \text{DegreesDepthCombination}(G_2, x_2)$ ▷ Get degree-depth combinations for G_2
return $\text{AreEquivalentLists}(Result_1, Result_2)$ ▷ Compare results (order-independent)

Additionally, during the research process, we need an algorithm to generate either a pair of isomorphic graphs or a pair of non-isomorphic but similar graphs for experimental purposes. The core steps of this algorithm are as follows:

1. Generate random graph G1 (Erdős-Rényi model)
2. If connectivity required, regenerate until connected
3. Create G2:
    - If isomorphic: copy G1
    - If non-isomorphic: modify k-couple edges (the number of edges cannot change)
    - (remove k existed, add k non-existed)
    - Check the non-isomorphism and regenerate if needed
    - (prevent the coincidental occurrence of isomorphism after modifying k pairs of edges)
4. Randomize G2's node ordering
5. Return G1, G2

Core params:

- n_nodes: number of nodes
- p_edge: probability of edge generation (used in Erdős-Rényi model)
- is_isomorphic: whether graphs should be isomorphic
- is_connected: whether graphs should be connected
- is_directed: whether graphs are directed
- k: if non-isomorphic, number of edge pairs to modify

4.3 Experiments

4.3.1 Selection of params

`is_connected=True`. Since connected graphs are the main research objects in graph isomorphism problems.

`is_directed=False`. Currently, I first attempt to implement this algorithm on undirected graphs, as connected undirected graphs possess strong connectivity, where a single source node can reach all other nodes.

`k=1`. To thoroughly test the algorithm's ability to distinguish subtle differences, each pair of non-isomorphic graphs under all parameters differs by only one pair of edges.

The remaining parameters are as follows, which are expected to demonstrate the experimental results of this algorithm on graphs of different sizes and densities.

n_nodes	p_edge			k
10	0.1	0.2	0.3	1
50	0.05	0.1	0.2	1
100	0.025	0.05	0.1	1
200	0.02	0.04	0.08	1
500	-	-	0.05	1

Figure 10:

4.3.2 Setup and notations

For each parameter combination, repeat the experiment 500 times.

Generate isomorphic or non-isomorphic graphs with equal probability.

Compare the algorithm results with the actual isomorphism.

Calculate the statistics for TP (True Positive), FP (False Positive), FN (False Negative), and TN (True Negative). (P- isomorphic N-non-isomorphic)

4.3.3 Experiment results

The fact that FN (False Negatives) consistently remains at 0 while FP (False Positives) is mostly non-zero indicates that all cases judged as non-isomorphic are correct, while there are some errors in cases judged as isomorphic.

$n = 10$: error rate 1%

$n \geq 50$: error rate 0.6%

This indicates that after a single BFS, the algorithm already achieves considerably good results in distinguishing between isomorphic and non-isomorphic graphs. The slightly higher error rate in smaller graphs can be attributed to their less rich depth information and fewer degree distribution patterns.

n_node,p_edge	TP	FP	FN	TN
10,0.1	252	4	0	244
10,0.2	260	6	0	244
10,0.3	254	5	0	241
50,0.05	255	2	0	243
50,0.1	259	2	0	239
50,0.2	234	3	0	263
100,0.025	261	1	0	238
100,0.05	256	0	0	244
100,0.1	258	2	0	240
200,0.02	253	2	0	245
200,0.04	260	2	0	238
200,0.08	244	1	0	255
500,0.05	259	1	0	240

Figure 11:

4.3.4 Supplementary Verification

The above results are not always satisfactory, as the error rate still needs to be further reduced. Therefore, repeated verification is considered.

Given G1 and G2, the algorithm CompareGraphStructures(G1, G2, x1, x2) can be repeatedly tested with different values of x. If G1 and G2 are isomorphic, the algorithm should return True for any choice of x. If G1 and G2 are non-isomorphic, starting BFS from nodes of different degrees is likely to capture new structural differences. Once the algorithm returns False for any value of x, we can definitively confirm that G1 and G2 are non-isomorphic.

Do supplementary verification for cases that FP>1:

- x1 = second_min_degree(G1)

- x2 = second_min_degree(G2)

For the new value of x, we prefer a significant difference from the original value. Since the minimum degree is 1, and sparse graphs have too many nodes with degree 1, we choose nodes with the second smallest degree as source nodes.

The results show that after adding one additional verification, except for the case of n=10 and p=0.3 (which represents the maximum p value in the experiment), the number of False Positives (FP) in all other graphs decreased to either 0 or 1, with an error rate 0.2%.

n_node,p_edge	TP	FP	FN	TN
10,0.1	250	1	0	249
10,0.2	249	0	0	251
10,0.3	256	2	0	242
50,0.05	261	0	0	239
50,0.1	238	1	0	261
50,0.2	249	1	0	250
100,0.1	245	0	0	255
200,0.02	265	1	0	234
200,0.04	246	1	0	253

Figure 12:

4.4 Algorithm Complexity Analysis

V: number of nodes

E: number of edges

Algorithm DegreesDepthCombination(G, x):

- Calculate degrees: O(V) (list) or O(V+E) (BFS)
- Initialization: O(V)
- Multi-source BFS:O(V+E)

CompareGraphStructures(G1, G2, x1, x2):

- DegreesDepthCombination: O(V+E)
- AreEquivalentLists: O(V) (Hash table)

Number of CompareGraphStructures(G1, G2, x1, x2): 1 or 2 for most cases, and will not increase with V and E

Overall: O(V+E)

4.5 Discussion

4.5.1 Application of the Algorithm in Directed Graphs

Strongly Connected Graph The algorithm demonstrates excellent performance due to the inherent properties of strongly connected graphs, where BFS can effectively capture the depth information of all nodes. However, strongly connected graphs represent just one well-behaved case; in practical applications, weakly connected graphs are far more prevalent.

Weakly Connected Graph Due to the limitation that a single BFS cannot traverse all nodes, the acquisition of depth information is severely constrained. Consequently, implementing the algorithm by first extracting strongly connected components followed by separate BFS operations may be necessary. Tarjan's Algorithm presents an efficient solution, offering linear-time complexity for strongly connected component extraction.

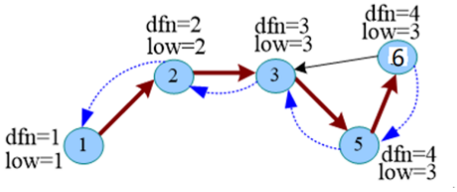


Figure 13:

4.5.2 Application of the Algorithm in Subgraph Isomorphism

For determining whether a complex structured graph exists within a larger graph:

When there are fewer nodes with maximum degree:

The auxiliary search is effective: The algorithm runs on all nodes in the larger graph whose degrees are the maximum degree of the smaller graph. By comparing these results with those from the smaller graph, a high local similarity suggests potential subgraph isomorphism. Statistical patterns indicate that having fewer nodes of maximum degree is the common case.

When there are many nodes with maximum degree:

The algorithm encounters huge computational complexity because there are too many cases to compare.

Additionally, it performs poorly when counting the occurrences of simple structured graphs in a larger graph, as simple graphs have limited depth information and degree distribution patterns, making discrimination difficult.

5 Summary

In this paper, we explored three techniques for GI pruning. The first technique is Monte Carlo approach based on Perron-Frobenius theorem. We tried particle simulation to estimate the Perron vector for pruning. The second technique is Lanczos pruning, motivated by the sensitivity and failure of Monte Carlo method. We used a orthonormal matrix Q_n to squeeze weighted adjacency matrix and by Cauchy interlacing theorem pruned the candidates with low time and space complexity. The third techniques is BFS pruning that take the degree and depth of each vertices as features to prune, utilizing more structural information of graph.

The three techniques originate from different fields. The Monte Carlo approach is from statistical computation which usually provides approximate results with error bound in term of probability. The Lanczos pruning is from matrix analysis and numerical linear algebra, which can be theoretically analyzed before experiments and provide precise results. But it sometimes failed due to numerical unstablity. The BFS pruning is from classical discrete mathematics which also provides precise results and can explore more structural features of graph that are invisible in adjacency matrix.

In the future, we recommand researchers interested in GI taking a wider and more multidisciplinary view on this question. For example, for the BFS pruning, can we apply Monte Carlo approach to futher reduce the cost of BFS and guarantee the correctness by a very high probability? For the Lanczos pruning, during the iteration, can we choose some well-designed orthonormal matrix $\tilde{Q}_n$ such that the distinguishable structural feature of adjacency matrix A can be revealed as early as possible? In the interests of brevity, such inquiries are best left for future research.

References

[1] Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, ii. *Journal of Symbolic Computation*, 60:94–112, 2014.

[2] Carsten Henneges, Christoph Behle, and Andreas Zell. Practical graph isomorphism for graphlet data mining in protein structures. In *Computational Intelligence: Revised and Se-*

lected Papers of the International Joint Conference, IJCCI 2010, Valencia, Spain, October 2010, pages 345–360. Springer, 2012.

[3] László Babai, William M Kantor, and Eugene M Luks. Computational complexity and the classification of finite simple groups. In *24th Annual Symposium on Foundations of Computer Science (Sfcs 1983)*, pages 162–171. IEEE, 1983.

[4] Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012.

[5] Lloyd N Trefethen and David Bau. *Numerical linear algebra*. SIAM, 2022.