Google Summer of Code 2024 Proposal
NetworkX | NumFocus

Incorporate a Pyhon library for ISMAGs isomorphism calculations

Mohamed Rezk
mohrizq895@gmail.com

April 2024

"I think Open Source is the right thing to do the same way I believe science is better than alchemy"

– Linus Torvalds

Table of Contents

1 Abstract

Informally, the Graph Isomorphism problem involves deciding whether two given graphs are structurally identical, or, in mathematical terms, isomorphic. The problem first originated in the field of chemistry to check if two molecular structures are isomorphic. In mathematical terms, we define a **graph isomorphism** from graph G to graph H as a **bijective, adjacency-preserving** function:

$$f : V(G) \mapsto V(H).$$

Although graph isomorphism is $\mathcal{NP}$-Complete, there are several algorithms that can reduce the work required. One of these algorithms is **The Index-Based Subgraph Matching Algorithm (ISMA)**. This algorithm can work well on small instances of subgraphs with simple cyclic rotation or reflection symmetries. Such symmetries take place between two nodes, for instance, switching 2 nodes in a ring. However, ISMA cannot handle complex symmetries involving the simultaneous switching of sets of nodes, such as those in a permuted Petersen Graph. An enhancement to the algorithm is introduced in **The Index-Based Subgraph Matching Algorithm (ISMA) with general symmetries (ISMAGS)**. It simply utilizes the symmetry between the nodes of the subgraph sg to reduce the search space and hence returns only one instance of the matching.

2 Technical Details

The project is about incorporating two implementations of the ISMAGS algorithm from NetworkX and Sandia Labs into the codebase of NetworkX. The main idea of the algorithm is to detect the inherent symmetric properties present in the subgraph structure and convert these properties into pruning rules or constraints to break such symmetries in order to reduce the search space and speed up the computations. This is done by the use of two algorithms - symmetry detection followed by symmetry breaking. After these steps, the classic ISMA algorithm can be executed to obtain the final isomorphism results.

2.1 Symmetry Detection Algorithm

2.1.1 Subgraph Partitioning

The first step in symmetry detection is to partition subgraph nodes sgn_i with similar properties, i.e., the number and types of incoming and outgoing edges. The nodes are grouped into an ordered partition,

$$\pi = [W_1|W_2|\cdots|W_n]$$

where each cell W_i contains the nodes with the same properties. Initially, all subgraph nodes sgn are in the same partition cell W_1, and then that cell is refined if contained nodes don't have the same properties. For instance, consider graph (a) in the following figure.

The graph contains 4 nodes with 3 types of edges namely X, Y, Z. At first, all subgraph nodes are in the same cell W_1 as illustrated in graph (b); however, it is obvious that node 2 and node 4 do not have the same properties due to incompatible edge types, and it is more obvious that nodes with the same colors can exchange places without changing the overall topology of the graph, so the cell W_1 is refined, and another cell is introduced, as shown in graph (c).

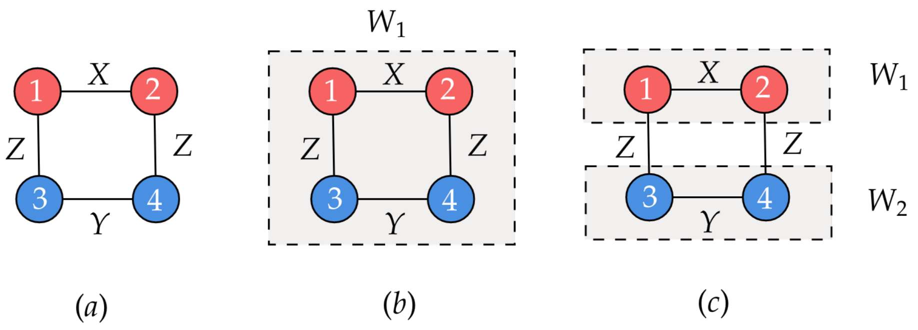

Figure 1: Figure showing the process of partition refinement

The process of refinement is repeated until we cannot make any progress.

2.1.2 Ordered Partition Pair & Recursive Refinement

In reality, the symmetry analysis of subgraph utilizes 2 partition sets namely the top partition π_t and the bottom partition π_b,

$$\pi_t = [T_1|T_2|\cdots|T_t], \quad \pi_b = [B_1|B_2|\cdots|B_b]$$

This pair is called **ordered partition pair**. Such pair will be used to detect the symmetries by recursively refining the coupling generated by traversing a search tree in a depth-first like manner. While traversing the search tree, orbits are generated, and branches are pruned to reduce the search space.

2.1.3 Orbit Pruning

Orbit pruning is an optimization technique to reduce the search space or prune branches in the search tree during the symmetry analysis, generating a set of permutation for the automorphism group A, which will act on the set of all possible permutations S of the subgraph. Given an element e in the set S, its orbit can be defined as the set of all elements of S that can be found by combining all permutations in A with e. Such orbits partition the set S as

$$S = \{O_1|O_2|\cdots|O_z\}$$

$$\forall e \in O_i, \forall P_k \in A : P_k(e) \in O_i$$

Generally, the orbit pruning is based on partitioning the set S, to reduce the search space, as one element suffices to generate the same orbit by applying a generating permutation. With orbit pruning, the algorithm exhausts all the permutations in the set S requiring a vast amount of computational power.

2.2 Symmetry-Breaking Algorithm

The symmetry detection algorithm generates a set of valid permutations that, when applied to a given instance of subgraph sg, outputs another sg that is isomorphic to the input instance. To avoid generating these isomorphic instances, a symmetry-breaking approach has to be introduced.

2.2.1 Stabilizers & Stabilizer Chains

Not all the permutations generated in symmetry detection change or swap all the nodes in the subgraph. This leads us to define stabilizer groups for every subgraph node sgn_i in sg as:

$$i_G = \{\ P_k \mid P_k(i) = i,\quad P_k \in G\ \},$$

which means that when a permutation P_k is applied to node i in a graph, the node stays in its respective place with no swapping, given that P_k belongs to some permutation group G. This notion can be repeated by what is called stabilizer chains, formally defined as

$$\forall i = 1 \dots s : G_i = i_{G_{i-1}}$$

In simpler terms, if we have a permutation group G_1 that is composed of a set of permutations, which do not swap node 1, then G_2 can be defined as the set of permutations in G_1 that do not swap node 2.

2.2.2 Coset Representatives

Once stabilizers are generated for every sgn_i, a mapping between nodes has to be established: cosets. The coset representative C_i of subgraph node sgn_i is defined as the set of subgraph nodes sgn_x that can be mapped to sgn_i in G_{i-1}, or more formally,

$$\forall i = 1 \dots s :\ C_i = \{\ sgn_x \mid P(sgn_i) = sgn_x,\ P \in G_{i-1}\}.$$

To generate the symmetry-breaking constraints, only coset representatives are needed; however, stabilizer chains G_i are not fully generated.

2.2.3 Generating Constraints

The coset representatives are converted into constraints on the IDs of the graph g nodes mapped to the subgraph sg nodes. For every subgraph node sgn_j in the coset C_i of subgraph node sgn_i, a constraint is generated to force the ID of the graph node mapped to subgraph node sgn_j to be higher than the ID of the graph node mapped to subgraph node sgn_i, more formally,

$$\forall i, j : sgn_j \in C_i, i \neq j \ : ID_i < ID_j$$

3 Schedule of Deliverables

3.1 Application Deadline & Community Bonding (April 2 - May 26)

During this period, I will ensure that my development environment is properly configured. Additionally, this time may include some extra contributions to NetworkX to become more familiar with the codebase and integrate into the community. These contributions may involve bug fixes, issue resolutions, code refactoring, or requests for new features or algorithms. Furthermore, I will discuss high-level questions with mentors and seek guidance on best practices and the overall architecture of NetworkX to ensure that the final code is properly written and documented. Finally, as NumFocus requires regular blog posts, I will create a base template for these posts. The website is already up and running here: The Computation Side of Things.

3.2 Phase 1 (May 27 - July 11)

3.2.1 Phase 1.1 (May 27 - June 7)

During this time, I will be taking my final exams for the spring semester. Unfortunately, I cannot do much, but I will continue to familiarize myself with the implementations of the algorithm. If I fall behind my plan, I will work on weekends to get back on track. Additionally, my fall semester begins in October, so if there is anything left, I will use that time (post GSOC until October) to catch up on what is missed.

3.2.2 Phase 1.2 (June 8 - June 15)

The main focus of this phase is to become more familiar with the ISMAGS algorithm by reading the research paper that introduced the algorithm. This will help me understand the intended performance of the algorithm. After that, I will read the documentation of both implementations of the algorithm (NetworkX and Sandia Labs), keeping in mind the differences in architectures and underlying data structures (NetworkX types).

By the end of this phase, I should have a good grasp of the theoretical aspect of the algorithm from the paper as a reference and both implementations. Additionally, I will discuss my findings with mentors for further refinement.

3.2.3 Phase 1.3 (June 16 - July 11)

The first 10 days of this phase will be dedicated to writing a solid merging plan. This plan will consist of several steps:

- Identifying common functionalities and differences between them.
- Thoroughly benchmarking both implementations to identify areas for improvement.
- Evaluating the scalability and maintainability of both implementations.
- Identifying shortcomings and bottlenecks in both implementations and searching for more optimized solutions.

The merge plan will be written as ordered steps to make the whole process easier to track and implement.

The remaining 2 weeks will be spent implementing the merging plan and ensuring that every step is going as planned. Additionally, I will open a pull request to track the merge process, detailing the thoughts and findings I encounter along the way.

3.2.4 Summary of this phase

- Make the underlying architecture of the algorithm more compatible with NetworkX
- Write a detailed merge plan
- Execute the merge plan
- Prepare for midterm evaluation

3.3 Phase 2 (July 12 - August 19)

3.3.1 Phase 2.1 (July 12 - August 4)

The first two weeks of this part of phase 2 will be dedicated to completing the incorporation plan, going back and forth between both implementations and the paper, testing on large instances, and making the best out of both implementations. Additionally, I will keep in mind the isomorphism API the algorithm has to provide, like `is_isomorphic(G1, G2)`. At this point, the algorithm should handle **directed graphs** isomorphism as the current implementation does not offer this feature.

The last week of this phase will be spent refining the integrated code by optimizing the performance where possible and addressing any issues or bugs discovered during testing. This will be ensured by using optimal data structures, and the final code should be readable and idiomatic.

Finally, the pull request tracking the integration process will be updated with feedback from mentors addressing any problems that could arise.

3.3.2 Phase 2.2 (August 5 - August 19)

This part of phase 2 will be mainly dedicated to the process of **documentation** and **testing**.

Documentation

The current NetworkX implementation of the ISMAGS algorithm is not completely documented. I will heavily document the algorithm, showing how it works while taking into account proper rST formatting for the Sphinx documentation generator. Additionally, I will add more doctests to the algorithm APIs. Furthermore, the documentation might include an explanation of why ISMAGS is superior compared to other algorithms. This might lead me to write a simple program or notebook to compare the graph isomorphism algorithms currently implemented by NetworkX. The program should benchmark the algorithms using some large instances of networks (possibly from Stanford Snap) and present the obtained results in terms of performance, memory, and scalability.

Testing

While the current NetworkX implementation of the ISMAGS algorithm is covered with some tests, the incorporation process may introduce new APIs and/or changes to existing ones, making it crucial to add new tests and modify existing ones. Furthermore, more tests for edge cases will be implemented to ensure good test coverage for the algorithm.

3.3.3 Summary of this phase

- Continue the merge plan
- Add support for directed graphs
- Refine API & optimize the final code
- Document the algorithm
- Add more tests for the algorithm and edge cases

3.4 Final Week (August 19 - 26)

By the beginning of the final week, I should have completed all the coding and development for the project. I will ensure that the project plan is met and that no important elements are overlooked in the process. Having an extra week for checking is definitely advantageous, as many issues can unexpectedly arise.

Moreover, at this point, I will prepare for the final evaluation by ensuring that the pull requests are adequately prepared for merging into the main branch, by obtaining feedback from mentors and addressing any minor bugs or recommended edits. This step will also ensure that the documentation and tests are properly completed.

4 Development Experience

I have coded many projects since my senior year of high school. Generally, I use Python as my primary language to model problems that interest me. For instance, I wrote a comprehensive article about the generalization of the fifteen puzzle and the proper algorithms that can be utilized to solve instances of the fifteen puzzle. In that model, I used Python to implement the **Weighted Iterative Deepening A*** algorithm with some heuristics to solve such puzzles. The code can be browsed here.

Two other projects that I am proud of are automata-cli and CV.py. **Automata-cli** is about manipulating programmatic specifications of automata. Such manipulations include (but are not limited to) rendering the specifications of the automata to comprehensive graphs via the Graphviz Python API, converting to other forms of automata (e.g., NFA to DFA), minimizing the automata to a minimal form, or generally executing any algorithm on the structure. The project is still under development and targets students and professors working in this field.

CV.py is a kind of wrapper on LaTeX. The primary idea is to convert the curriculum vitae (CV) information written in a simple YAML file to a PDF through a LaTeX cloud compiler. The user does not have to know the cumbersome LaTeX syntax or even have a LaTeX compiler locally installed on their machine. The main focus lies in the abstract text written in the file, with no special formatting needed as this is handled by predefined templates.

In college-related projects, I have used C/C++ to implement things like the Lampel-Ziv algorithm and GPS tracking algorithm with shortest path on an embedded board. With Python, I have written some quantum algorithms with the **QisKit** library and many other projects like the Gobblet AI player with MINMAX algorithm. Additionally, I use shell scripting to automate some Linux tasks like bookmarks.sh and I3 blocks scripts.

5 Why This Project?

There are plenty of reasons why this project is compelling to me. They are listed as separate points to make the view clearer.

Graph Isomorphism Problem:

The general problem of **graph isomorphism** is extremely useful in many fields, especially chemistry and biology. Graph isomorphism is used left and right to analyze **protein structures**, **molecular structures and chemical bonds**, **social networks**, **automata theory**, **compiler optimization**, etc. The applications are countless.

The ISMAGS Algorithm:

The ISMAGS algorithm presents a significant advancement in the area of subgraph matching and network analysis. By integrating internal symmetries of subgraphs as constraints, ISMAGS reduces search space and query times substantially, making it highly efficient even for large networks with multiple edge types. Its approach to symmetry analysis and constraint generation minimizes unnecessary computations. Moreover, ISMAGS demonstrates superior performance compared to existing algorithms like **VF2** and **ISMA** in terms of both time and space complexity, making it a promising avenue for further research.

NetworkX:

Based on my experience, NetworkX maintains transparent communication channels and actively engages with its contributors, facilitating knowledge sharing and mentorship opportunities. By being part of the NetworkX community, contributors not only have the chance to shape the development of a widely-used library but also to make meaningful connections with like-minded individuals passionate about leveraging networks to solve real-world problems. Additionally, the quality and readability of the NetworkX codebase are far superior; the code is written in an idiomatic manner, which is the style I most admire. Moreover, the adapted development workflow is intuitive and simple to maintain.

6 Personal Information

Name: Mohamed Mahmoud Rezk

Preferred Name: Rezk

Email: mohrizq895@gmail.com

GitHub: mohamedrezk122

Website: The Computation Side of Things

I am a pre-final year undergraduate student at Ain Shams University, Cairo, Egypt, majoring in Electrical and Computer Engineering. Primarily, I am interested in theoretical computer science and mathematics. This interest drove me to study the fascinating area of graph theory, where I wrote my first algorithm to solve arbitrarily large mazes four years ago. Ironically enough, I utilized NetworkX in that project to store the graph structure, marking my first exposure to this library. After engaging in several projects, research experiences, and readings, I am planning to pursue a PhD in theoretical computer science with a focus on graph algorithms. Graphs can model various aspects of our lives, not just abstract mathematical structures, making development in this area crucial as well as enjoyable.

6.1 Miscellaneous

6.1.1 Operating System

I use **Void Linux** as my daily driver with **i3 tiling window manager** and some custom configurations. My terminal is **simple terminal (st)**, which is written by the **Suckless** group. Also, I use **ZSH** with **oh-my-zsh** as an interactive shell, and **dash** as a Posix shell.

6.1.2 Text Editor

Generally, I use **Sublime Text** for Python, LaTeX, C++, and any other heavy-duty tasks. Sublime is fast, lightweight, and usable out of the box, unlike GNU Emacs. The only drawback with Sublime is that it is not open-source. However, sometimes I use **GNU Emacs** for Org-Mode, mainly for writing college reports. Moreover, I use **LunarVim** (an IDE layer for NeoVim) for quick tasks like shell scripting. Lately, I have been considering switching to LunarVim completely due to the myriad of plugins available for NeoVim.

7 References

- The Index-Based Subgraph Matching Algorithm with General Symmetries (ISMAGS): Exploiting Symmetry for Faster Subgraph Enumeration
- The Index-Based Subgraph Matching Algorithm (ISMA): Fast Subgraph Enumeration in Large Networks Using Optimized Search Trees
- NetworkX/ISMAGS Algorithm
- Sandia Labs/ISMAGS-in-Python