SCALING UP SUBGRAPH ISOMORPHISM

Cheong Sik Feng[1], Kong Xin Yang[1], Chieu Hai Leong[2], Wong Jialiang Joshua[2]

[1]NUS High School of Mathematics and Science, 20 Clementi Avenue 1 Singapore 129957
[2]DSO National Laboratories, 12 Science Park Drive Singapore 118225

ABSTRACT

In chemical structure analysis, we need to apply subgraph isomorphism (SGI) algorithms to compare a large set of substructures to another large set of candidate structures. Since SGI is NP-complete and computationally expensive, it has a long run time. In this project, we aim to speed up the process of SGI, and as a result speed up chemical structure elucidation, via eliminating non-subgraphs immediately through the establishment of graph signatures. To aid in our experiment, we have been provided with a set of multiple subgraphs and graphs as data to be inputted into our code. We evaluated three software toolkits igraph, RDKit and CDK, as well as the signature tools in these software such as the BitFingerprint in CDK. We developed a fingerprint that combines the BitFingerprint with counts of atoms and showed that this combination outperforms all other methods in reducing the run time of our experiments.

INTRODUCTION

Goal

The goal of this project is to reduce the time taken for subgraph isomorphism, which can be used to subsequently reduce the time taken for chemical structure elucidation (CSE).

Literature Review

In this report, we define CSE as the problem of elucidating the structure of an unknown chemical compound given its mass spectrum and the number of each atom type that are present in the compound. The application of CSE can be clearly illustrated using the following example. Consider a scenario where a country has been attacked by terrorists, via an unknown chemical weapon targeted at the civilians. In order to counteract this deadly weapon and minimize casualties, a plan to neutralize this deadly chemical must be implemented. The chemical structure of the weapon is essential to figuring out how to counter it, which is where CSE comes in. Using a mass spectrometry histogram of the substance, a skilled chemist can figure out the identity of this unknown substance in a few days. However, in these few days, the chemical weapon would have already affected many civilians. Therefore, the speeding up of CSE is essential for reducing the adverse effects this can have on a nation, particularly the citizen's feelings of security and trust in the country, which may result in the crippling of a nation.

Not only do scientists want to make CSE faster but also completely autonomous [1]. The faster process will vastly decrease the time taken for CSE. The complete autonomous process will allow a skilled chemist to identify the chemical weapon in the shortest time. Doing CSE autonomously requires checking of SGI over huge datasets, which in turn will drastically increase the run time for our program. Given m graphs and n subgraphs, SGI is not checked once on every graph and possible subgraph (m+n times), but it is checked for each unique pair of a graph and subgraph

(m*n times). In order to speed this up, we want to eliminate some graph and subgraph pairs via certain conditions. For example, if the subgraph has a larger number of atoms or certain groups of atoms than the graph itself, then that pair can be immediately eliminated as not possible.

The subgraph isomorphism (SGI) problem is a computational task where one must determine whether there exists a subgraph in a graph G that is isomorphic to another graph H. SGI is known to be NP-complete (non-polynomial deterministic time complete), and its worst-case time complexity is O(V!×V) using the VF2 algorithm [7], where V is the number of vertices. An NP-complete problem does not have a known algorithm that solves it in polynomial time.

Using a large number of SGI computations for chemical structure elucidation (CSE) from mass spectrometry by matching substructures [1], the main bottleneck is SGI. Our project aims to improve the processing speed for CSE by reducing the run time required for SGI.

In this research paper, we will be further discussing the conditions for the elimination of graph-subgraph pairs, breaking down what our program does as well as the rationale between choosing which language to implement in our program, to ultimately reduce the number of times SGI is checked, which is the bottleneck in CSE [1].

MATERIALS AND METHODS

Libraries

- igraph 0.7.1 with Python 3.7.3
- CDK2.2 with OpenJDK Runtime Environment (build 1.8.0_222-b05)
- RDKit 2019.03.1 with Python 3.7.3
- RDKit 2018.09.1 with GCC 9.1.0

igraph is a library for creating and manipulating graphs. It is intended to be as powerful (i.e. fast) as possible to enable the analysis of large graphs. It was written in C and with a Python API.

Chemistry Development Kit (CDK) is a collection of modular Java libraries for processing chemical information (Cheminformatics).

RDKit is a collection of cheminformatics and machine-learning software written in C++ and Python. The core data structures and algorithms are written in C++, while the Python 3 wrappers are generated using Boost.Python.

The programs were run on an Intel i5-2410M CPU.

Dataset

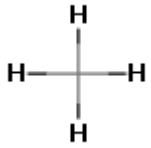
We measured our programs on datasets provided by our mentors. The datasets contain structures of graphs and subgraphs of phosphonothionates.

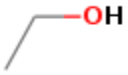
Methods

For our research, the objective is to find faster ways of doing SGI, experimenting with different libraries and different graph signatures.

Test 1 - Comparison with and without atom count signatures for SGI

We run a program that does SGI with a simple graph signature and another without graph signatures using igraph, and we compared their run time. The graph signature we used here counts the number of atoms of each element in the graph. All subgraphs must have fewer or equal number of atoms of each element. We measure how long it takes to find the number of graph-subgraph pairs in the dataset.





Methane *Ethanol*

Molecule structures are taken from the NIST Chemistry Webbook [7]

For example, methane has only 1 carbon atom, while ethanol has 2 carbon atoms, 1 oxygen atom, and 1 hydrogen atom. Since there are more carbon, oxygen and hydrogen atoms in ethanol than methane, methane is a possible substructure of ethanol, and it is indeed a substructure. Do note that we do not consider implicit hydrogens, only the explicit hydrogens, such as the hydrogen in the hydroxyl group are considered. We compared the run time and number of SGI operations with and without graph signatures in Table 1 in the Appendix. From this test, we conclude that the graph signature does indeed reduce the time taken for SGI, as well as the number of times the SGI algorithm is run.

Test 2 – Comparison of SGI using igraph and CDK.

We compare the run time of igraph and CDK on two datasets. The first dataset comprises of 293 graphs and 34979 subgraphs, and the second dataset comprises of 1713 graphs and 173584 subgraphs. Both igraph and CDK were first run without any graph signatures, and subsequently with the graph signature that we had used in Test 1. From this test, CDK is significantly faster than igraph, hence we decided to proceed with comparing our newer methods against CDK.

Test 3 – Comparison of SGI using different graph signatures

We test several different types of graph signatures, such as counting the types of atoms, counting the number of certain chains, and generating a BitFingerprint.

A BitFingerprint is an array of bits, where each bit represents certain information about the molecule. The fingerprint that we used for CDK contains 1024 bits, where each bit represents whether certain paths of atoms are present [4]. A subgraph should never have any path not present in the graph.

After running these tests, we had an idea to merge the different methods together by implementing a "CountFingerprint". The CountFingerprint, like the BitFingerprint, is an array of integers, where each integer represents the number of times certain paths of atoms are present in the molecule. The difference is that the CountFingerprint also would count the number of occurrences of the particular path. Such an implementation does increase the speed of the algorithm, but also increases the memory used noticeably.

Test 4 – Comparison of SGI with graph similarity signatures

We attempted to compute graph similarities using the Tanimoto Similarity between the fingerprints of the graphs and subgraphs. If the similarity coefficient was above some value, we would then consider it similar enough to be worth doing SGI. We measured the time taken and percentage accuracy for several cutoff values. However, this method is more probabilistic, as there is always a chance that the subgraph is not similar to the graph but still a subgraph nonetheless, especially if the subgraph is a lot smaller than the graph.

Test 5 – Comparison of SGI using RDKit (C++ and Python) and CDK

We compared the time taken and the number of SGI operations with RDKit using C++ and Python, using the most efficient combination of graph signatures found in the comparison of SGI using different graph signatures.

RESULTS

Table 1: comparison of run time and number of SGI operations required with and without the use of graph signatures. All run time is reported using igraph.

# of Graphs	# of Subgraphs	Direct		With graph signatures	
		Run time (s)	# SGI operations	Run time (s)	# SGI operations
293	343	2.86	100,499	1.64	64,405
1715	2126	121.29	3,646,090	64.97	2,836,830

Table 2: comparison of run time and number of SGI operations with and without the use of graph signatures. Run time is reported using CDK and igraph

Graphs	Subgraphs	CDK		IGRAPH	
		Direct	With graph signature	Direct	With graph signature
293	34,979	38.39s 10,248,847 SGI	166.26s 8,702,199 SGI	273.44s 10,248,847 SGI	166.26s 8,348,655 SGI
1713	173,584	1563.34s 297,349,392 SGI	1109.78s 208,905,713 SGI	10642.47s 297,349,392 SGI	5956.43s 208,068,755 SGI

Table 3: comparison of run time and number of SGI operations with different graph signatures. All run time is reported using CDK

- *H0 – Direct*
- *H1 – Count atoms*
- *H2 – Count chains*
- *H3 – BitFingerprint*
- *H4 – CountFingerprint*

	293 graphs 34,974 subgraphs	1713 graphs 173,584 subgraphs
H0	38.90s 10,248,847 SGI	1474.87s 297,349,392 SGI
H1	32.231s 8,702,199 SGI	1031.62s 208,905,713 SGI
H2	36.96s 8,354,016 SGI	1671.23s 296,942,352 SGI
H3	9.11s 252,469 SGI	186.61s 7,895,048 SGI
H4	6.34s 192,383 SGI	103.71s 4,681,512 SGI
H1+H2	29.175s 7,125,558 SGI	1218.80s 208,905,713 SGI
H1+H3	8.476s 245,219 SGI	165.954s 7,845,089 SGI
H1+H4	6.642s 192,383 SGI	93.82s 4,681,512 SGI
H2+H3	9.766s 252,469 SGI	184.465s 7,895,044 SGI
H2+H4	6.923s 192,383 SGI	111.195s 4,681,512 SGI
H1+H2+H3	9.211s 245,219 SGI	173.895s 7,845,089 SGI
H1+H2+H4	7.103s 192,383 SGI	98.886s 4,681,512 SGI

Table 4: comparison of using graph similarity as a graph signature

Threshold	Time	Subgraph – graph pairs found	Percentage missed
0.25	20.45s	129,833	21.4%
0.20	28.15s	139,809	13.4%
0.15	33.48s	149,786	7.27%
0.10	39.60s	157,005	2.80%
0.05	43.86s	161,189	0.206%

Table 5: comparison of run time and number of SGI operations required with and without the use of graph signatures. All run time is reported using CDK and RDKit

Data-set no.	Candidates	CDK (Java) Direct	CDK (Java) With graph signatures	RDKit (C++) Direct	RDKit (C++) With graph signatures	RDKit (Python) Direct*
112	44	4.51s 1,539,076 SGI	0.619s 2,412 SGI	1.98s 1,539,076 SGI	2.31s 2,395 SGI	5.24s 1,539,076 SGI
991	148	14.34s 5,176,892 SGI	1.594s 8,805 SGI	8.39s 5,176,892 SGI	3.54s 8,739 SGI	16.78s 5,176,892 SGI
3505	1584	147.76s 55,406,736 SGI	13.83s 105,411 SGI	72.85s 55,406,736 SGI	43.81s 106,774 SGI	83.37s 55,406,736 SGI
3599	2988	270.55s 104,517,252 SGI	26.27s 261,110 SGI	174.77s 104,517,252 SGI	88.12s 270,678 SGI	174.12s 104,517,252 SGI

**RDKit python was too slow and had to be run on 8 threads concurrently. Timings show time taken for 8 threads to complete SGI*

Table 6: comparison of run time for CSE using different libraries (Credit Mr. Chua Jing Yang)

Experiment	1	2	3	4	5
Package Used for SGI	iGraph	RDKit	RDKit	CDK	CDK
Cores	8	8	8	8	8
Chunk Size	250	250	numCand//8	250	numCand//8
Average Time Taken for Retrieval and Preparation of Data	437s	~340s	~340s	~340s	~340s
Average Time Taken for NN	< 0.1s	< 0.1s	< 0.1s	< 0.1s	< 0.1s
Average Time Taken for SGI and Scoring	496s	157s	190s	Takes too long, did not test for all test cases	91s
Total Average Time Taken	**933s**	**497s**	**530s**	-	**431s**

DISCUSSION

We have confirmed that using graph signatures will reduce the time needed for SGI. The fastest method is by using graph signatures with CDK (Java). The graph signatures that are used counts the number of atoms for each element and generates a CountFingerprint.

The CDK library is written in Java, while igraph is written in C with python bindings. SGI in CDK is much faster than in igraph. There are 2 possible reasons for this. Firstly, it may be due to some differences in the implementation, as a graph in CDK is a valid molecule which satisfies certain constraints, while in igraph there are less constrains to what may be considered a graph. Secondly, this may be simply due to Java being generally faster than Python as a programming language.

Using graph similarity can be faster, but it will miss out on some matches, and hence sacrifice its accuracy. A likely reason would be because some subgraphs may be very small compared to the size of the graph, hence the similarity function finds that it is rather dissimilar, and as a result incorrectly rejects it.

From the results, it shows that when doing SGI without graph signatures, RDKit (C++) is actually much faster than CDK (Java). This is likely due to the different implementations of the fingerprints, where the RDKit (C++) fingerprints are slower than the fingerprints in CDK (Java).

In brief, the ranking of the methods are:

1. CDK (Java) with graph signatures
2. RDKit (C++) with graph signatures
3. RDKit (C++) without graph signatures
4. CDK (Java) without graph signatures

FUTURE WORK

For SGI, further research can be done on implementing CDK's fingerprint implementation in RDKit (C++), which could potentially make RDKit (C++) faster than CDK (Java).

On the CSE problem, given the mass spectrometry of the molecule we want to identify, we can attempt to reduce the number of substructures through eliminating those substructures that are not represented in the mass spectrometry. We could try to compare the isotopic abundance of the substructure with the mass spectrometry to determine their presence. It is likely that many substructures will be represented in the same mass spectrometry. Many of them have a similar isotope abundance. This makes it difficult to be able to accurately deduce which substructures are even present in the mass spectrometry without the help of further advanced chemistry theory. Directly reducing the run time of CSE, instead of SGI, by reducing the number of substructures that need to be checked by using the mass spectrometry can be a future work of research.

CONCLUSION

We have confirmed that using graph signatures will reduce the time needed for SGI. The fastest method is by using graph signatures with CDK (Java). The graph signatures used counts the number of atoms for each element and generates a CountFingerprint.

Mr. Chua Jing Yang, DSO – NUS, has used our research program in his test cases of phosphonothionates. It has shown that the time taken for SGI and scoring is much faster than previous methods.

This project has achieved the goal to reduce the time taken for subgraph isomorphism, which does reduce the time taken for chemical structure elucidation.

ACKNOWLEDGEMENTS

We would like to thank our project mentors, Dr. Chieu Hai Leong and Mr. Wong Jialiang Joshua for their guidance throughout the process of our research. Mr. Chua Jing Yang and Mr. Eric Tan Lee Han of DSO National Laboratories have also provided us with much invaluable help.

We would also like to thank our teacher-mentors, Mr. Lim Teck Chow and Mr. Low Fook Hong, for supporting us throughout our SMP Journey.

REFERENCES

[1] Jing Lim, Joshua Wong, Minn Xuan Wong, Lee Han Eric Tan, Hai Leong Chieu, Davin Choo, Neng Kai Nigel Neo — Chemical Structure Elucidation from Mass Spectrometry by Matching Substructures. https://arxiv.org/abs/1811.07886. Last retrieved 17 June 2019.

[2] Gregory Landrum (2012) — RDKit Fingerprints. https://www.rdkit.org/UGM/2012/Landrum_RDKit_UGM.Fingerprints.Final.pptx.pdf. Last retrieved 15 June 2019.

[3] Egon Willihagen, John Mayfield, Rajarshi Guba, Christoph Steinback — Chemistry Development Kit. https://cdk.github.io/. Last retrieved 22 May 2019.

[4] Steinbeck et al. The Chemistry Development Kit (CDK): An Open-Source Java Library for Chemo- and Bioinformatics. J. Chem. Inf. Comput. Sci. 2003 Mar-Apr; 43(2):493-500, doi:10.1021/ci025584y (retrieved July 28, 2019)

[5] Similarity between structures. https://ctr.fandom.com/wiki/Report_the_similarity_between_two_structures. Last retrieved 27 June 2019.

[6] Roger Sayle — Improved SMILES substructure searching. http://www.daylight.com/meetings/emug00/Sayle/substruct.html. Last retrieved 21 June 2019.

[7] L.P. Cordella, P. Foggia, C. Sansone, M. Vento, (2004). A (sub)graph isomorphism algorithm for matching large graphs. IEEE Transactions on Pattern Analysis and Machine Intelligence, 26(10), 1367-1372. doi:10.1109/tpami.2004.75 (retrieved July 28, 2019)

[8] P.J. Linstrom and W.G. Mallard, Eds., NIST Chemistry WebBook, NIST Standard Reference Database Number 69, National Institute of Standards and Technology, Gaithersburg MD, 20899, doi:10.18434/T4D303, (retrieved July 28, 2019)

APPENDIX

All codes written for Test 1 to 5 can be found at https://gitlab.com/sikfeng/scaling-up-subgraph-isomorphism