Billion-scale Detection of Isomorphic Nodes

Luca Cappelletti
Department of Computer Science
University of Milan
Milan, Italy
0000-0002-1269-2038

Tommaso Fontana
Department of Computer Science
University of Milan
Milan, Italy
0000-0002-9806-3493

Justin Reese
EGSB
Berkeley Lab
Berkeley, USA
0000-0002-2170-2250

David A. Bader
Department of Data Science
New Jersey Institute of Technology
New Jersey, USA
0000-0002-7380-5876

***Abstract*—This paper presents an algorithm for detecting attributed high-degree node isomorphism. High-degree isomorphic nodes seldom happen by chance and often represent duplicated entities or data processing errors. By definition, isomorphic nodes are topologically indistinguishable and can be problematic in graph ML tasks. The algorithm employs a parallel, "degree-bounded" approach that fingerprints each node's local properties through a hash, which constrains the search to nodes within hash-defined buckets, thus minimising the number of comparisons. This method scales on graphs with billions of nodes and edges. Finally, we provide isomorphic node oddities identified in real-world data.**

***Index Terms*—graph, node isomorphism, parallel algorithm**

I. Introduction

Isomorphic graphs are a fundamental concept in graph theory, i.e. graphs that can be transformed into one another through a bijective node re-labelling function called graph isomorphism. There are no known polynomial time algorithms for detecting graph isomorphism; therefore, detecting them is computationally expensive [1]. This paper focuses on the more tractable problem of detecting attributed isomorphic node groups (INGs) within a graph. Nodes in an ING are topologically identical, as they cannot be distinguished based on their connections to other nodes in the graph (see Figure 1). Intuitively, swapping nodes within INGs does not change the graph topology. INGs can reveal peculiar node properties or quality control (QC) issues in real-world graphs. They may represent the same duplicated entity or elements that have become topologically identical due to data processing errors or lack of information. For example, in protein-protein interaction graphs such as STRING's, INGs may hint at the presence of isoform proteins [2]: generally, only one representative isoform protein is kept in the graph; therefore, detecting isoform groups would hint to preprocessing issues. By definition, nodes in INGs are topologically indistinguishable and, therefore, can be problematic in graph ML tasks.

In the literature, there exist *fuzzy* approaches to detect *semantically* duplicated nodes using their metadata, generally characterized by a quadratic complexity with respect to the number of nodes [3]. We present a novel parallel *exact* algorithm for detecting INGs in graphs with billions of attributed nodes and edges on a commodity desktop with, except for pathological cases, *linear* complexity with the number of nodes.

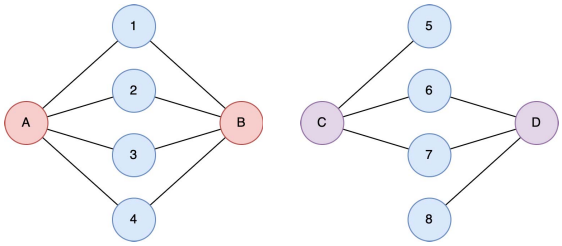

Fig. 1. **Isomorphic and non-isomorphic nodes:** the nodes A and B are isomorphic as they share all the neighbours and are topologically interchangeable. C and D are non-isomorphic, as they have a distinct topology.

Our algorithm is implemented in Rust with Python bindings as part of the open-source GRAPE graph processing and ML library [4]. The experiments code is available on GitHub [1].

II. Isomorphic Nodes

A graph $G = (V, E)$ (or multigraph) is constituted of a set of nodes V and a set of edges E. A node $v \in V$ has neighbours $\mathcal{N}(v)$ and degree $d(v) = |\mathcal{N}(v)|$.

Two nodes $a, b \in V$ are isomorphic if they have the same neighbours, except for $\{a, b\}$ themselves, which we denote as $\widehat{\mathcal{N}}_w(v) = \mathcal{N}(v) \setminus \{v, w\}$. Furthermore, if $a \in \mathcal{N}(b)$, then $b \in \mathcal{N}(a)$ and vice-versa. Therefore, we can define the equivalence relationship between a and b as:

$$a \cong b \iff (\widehat{\mathcal{N}}_b(a) = \widehat{\mathcal{N}}_a(b)) \wedge (b \in \mathcal{N}(a) \iff a \in \mathcal{N}(b))$$

Additional properties, such as edge labels and weights, should be compared when available.

III. Degree-bounded approach

Real-world graphs tend to have many low-degree nodes and a small number of high-degree nodes, following a scale-free degree distribution [5]. Given this distribution, it is improbable that two randomly chosen high-degree nodes would have the same neighbours. The probability of two nodes having the same neighbours decreases significantly with the degree of those nodes. Therefore, if two high-degree nodes have the same neighbours, they may be an ING oddity and warrant further investigation. This property can be exploited to reduce

[1] https://github.com/AnacletoLAB/grape/blob/main/tutorials/Billion-scale%20attributed%20isomorphic%20nodes%20with%20GRAPE.ipynb

979-8-3503-1199-0/23/$31.00 ©2023 IEEE
DOI 10.1109/IPDPSW59300.2023.00046

the number of nodes to be examined by restricting the search to nodes $v \in V$ with degree $d(v) \geq d_{\text{min}}$.

IV. Isomorphic Node Groups Detection

Checking exhaustively whether any two nodes $v, w \in V$ are isomorphic is expensive. Therefore, to scale, we must reduce the number of comparisons as much as possible. Our approach fingerprints each node's local properties through a hash: this allows us to limit the comparisons to the nodes within the same hash-defined bucket and to minimise the number of extensive comparisons to execute.

The algorithm starts by computing in parallel the vector of the nodes $v \in V$ that have a degree $d(v)$ higher than a specified minimum degree d_{min} (lines 3 through 5), alongside with a hash of the local properties of the node, such as the label and neighbours, using a provided hash method $\eta : V \to \mathbb{N}$. Our approach is detailed in Section V.

The algorithm then semi-sorts in parallel $\boldsymbol{H}$ (line 6) to create contiguous groups of nodes with identical hashes [6]. While sub-optimal, we employed a parallel BlockQuicksort [7] in the experiments. Using a semi-sort may lead to better performance.

We iterate in parallel over the contiguous groups of $\boldsymbol{\gamma} \in \boldsymbol{H}$ with identical hash. We start by creating the set of candidate isomorphic groups $\boldsymbol{I}'$ (line 9) and iterating on the nodes in $v \in \boldsymbol{\gamma}$, and for each one, we check whether v is isomorphic to any of the budding INGs $\boldsymbol{\gamma}' \in \boldsymbol{I}'$. We use a slight abuse of notation $v \cong \boldsymbol{\gamma}'$ to mean that we compare the node v with a node of $\boldsymbol{\gamma}'$. If we identify a compatible group, we add the node to it. Otherwise, we create a new singleton group with exclusively the node v (lines 10 to 19). Finally, we add to the INGs set $\boldsymbol{I}$ all of the non-trivial groups $\boldsymbol{\gamma}' \in \boldsymbol{I}'$ (lines 20 to 22).

V. Hashes

The hash strategies should produce for each node a hash that captures the local properties of a given node to minimise the number of exhaustive comparisons to execute and must not cause false negatives. The key insight is that two topologically indistinguishable isomorphic nodes $a \cong b$ can have different neighbourhoods $\mathcal{N}(a) \neq \mathcal{N}(b)$. Both nodes might appear in each-others neighbourhoods, e.g. $a \in \mathcal{N}(b)$, or have self-loops, e.g. $a \in \mathcal{N}(a)$. For this reason, to design a viable node-specific hash function $\eta : V \to \mathbb{N}$ such that $a \cong b \implies \eta(a) = \eta(b)$, it is paramount to identify node properties that avoid including all nodes $c \in V$ that **might be** isomorphic $a \cong c$ in their computation, while trying to minimise collisions.

A. Self-loops-excluded degree-based hash

Nodes may have one or more self-loops and be connected (Figure 2). Such nodes are isomorphic since they have the same topology. Yet, they do not share the same neighbourhoods.

Given a node v, we define its *self-loops-excluded degree* $\bar{d}(v)$ as the degree $d(v)$ minus the number of its self-loops:

$$\bar{d}(v) = d(v) - |\{w \in \mathcal{N}(v) \mid w = v\}|$$

Algorithm 1: Isomorphic node groups detection

Input : Graph $G = (V, E)$, degree d_{min}, hash $\eta : V \to \mathbb{N}$
Output: List of isomorphic node groups $\boldsymbol{I}$

```
1  I ← empty vector;
2  H ← empty vector;
3  for node v ∈ V do in parallel
4  |  if d(v) ≥ d_min then
5  |  |  H.append((η(v), v));
6  H ← semi-sort in parallel H;
7  H_iter ← iterator over slices of H with equal hash;
8  for candidate group γ ∈ H_iter do in parallel
9  |  I' ← [];
10 |  foreach node v ∈ γ do
11 |  |  match_found ← false;
12 |  |  foreach group γ' ∈ I' do
13 |  |  |  if v ≅ γ' then
14 |  |  |  |  match_found ← true;
15 |  |  |  |  γ'.append(v);
16 |  |  |  |  break;
17 |  |  if not match_found then
18 |  |  |  I'.append({v});
19 |  |  |  break;
20 |  foreach group γ' ∈ I' do
21 |  |  if |γ'| > 1 then
22 |  |  |  I.append(γ');
23 return I;
```

All isomorphic nodes have the same self-loops-excluded degree. Thus, we can use it as a component for the hash. The node label may be used in the hash if available.

B. Adjusted neighbours

We can improve the hash by including part of the immediate node neighbourhood. Given two nodes A and B, the subset of shared neighbours that are not the isomorphic nodes themselves is all nodes with self-loops-excluded degrees different from $\bar{d}(v)$ (Figure 2). For any node $v \in V$, we define an adjusted neighbourhood $\bar{\mathcal{N}}(v) \subseteq \mathcal{N}(v)$ as follows:

$$\bar{\mathcal{N}}(v) = \{w \in \mathcal{N}(v) \mid \bar{d}(v) \neq \bar{d}(w)\}$$

We can use the first k sorted adjusted neighbours as an additional hash component. The edge labels may be used in the hash if available. We exclude edge weights from the hash as they are subject to float errors which might cause false negatives.

VI. Experiments

The experiments were run on an *AMD Ryzen 9 3900x CPU 12 cores (24 threads) @ 4Ghz* paired with four banks of *32GB DDR4 3200 MT/s RAM (128GB)*. In all experiments, where not otherwise specified, we used a minimum degree of $d_{min} = 100$, $k = 1000$ adjusted neighbours, and as hash function AHash.

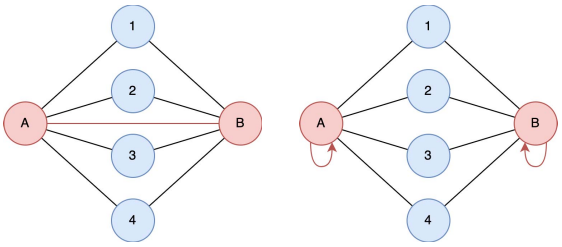

Fig. 2. **Connected isomorphic nodes with self-loops:** isomorphic nodes may be connected or have self-loops. Directly comparing their neighbourhood would fail, and more complex comparative schemas are necessary.

TABLE I
SUMMARY OF THE DATASETS' MAIN CHARACTERISTICS

Graph id	Graph name	Nodes		Edges	
		#	Types	#	Types
1	Saccharomyces Cerevisiae [2]	$7K$	$23K$	$1M$	✗
2	Homo Sapiens [2]	$20K$	$62K$	$6M$	✗
3	Mus Musculus [2]	$22K$	$57K$	$7M$	✗
4	KGCOVID19 [8]	$570K$	20	$18M$	47
5	Friendster [10]	$65M$	✗	$1.8G$	✗
6	Wikidata [9]	$1.3G$	✗	$6.2G$	$9K$
7	ClueWeb09 [10], [11]	$1.6G$	✗	$7.8G$	✗

A. Datasets

We considered seven real-world graphs. These datasets span from STRING's v11.5 weighted protein-protein interaction graphs [2] to knowledge graphs (KG) such as KGCOVID19 (v20221102) [8] and Wikidata (latest-truthy, 27-10-2021) [9], and web graphs such as Friendster [10] and ClueWeb09 [10], [11]. Whenever node or edge attributes are available, we employ them in node isomorphism detection. The datasets' main characteristics are summarised in Table I. The first column of Table I is the graph ID, and it is used in all other result tables.

B. Impact of hash function

We used four hash functions to evaluate the algorithm's sensibility to the used hash. These included xXhash [12], AHash [13], Siphash2-4 [14], and lastly, a simple custom hash composed of xor and the addition of a constant. We observe that the algorithm's performance is not sensitive to the hash uniformity guarantees and achieves the best performance with the simple hash, which has the least computational requirements and the no uniformity guarantee. The single thread wall times are in Table II.

C. Scalability

The algorithm generally shows near-linear scalability with the number of employed cores from 1 to 12. However, both for large graphs and hyper-threading, we observe marginal performance improvements; this may be caused by the algorithm being primarily memory-bounded by the memory accesses needed to compute the hashes. The wall times by the number of threads are in Table III.

TABLE II
SINGLE THREAD TIME (MS) BY HASH

Id	Simple	AHash	SipHash2-4	xXhash
1	25.3 ± 0.8	27.4 ± 0.6	44.7 ± 0.8	139 ± 4.5
2	136 ± 3	149.7 ± 1.3	242 ± 1.8	747 ± 15
3	148 ± 4	163.6 ± 2.7	266 ± 4.7	816 ± 22
4	$2k \pm 65$	$2k \pm 65$	$2.5k \pm 81$	$3.4k \pm 98$
5	$143k \pm 722$	$144k \pm 647$	$243k \pm 1.5k$	$344k \pm 2k$
6	389 ± 427	$420k \pm 873$	$446k \pm 962$	$586 \pm 1k$
7	$122k \pm 685$	$126k \pm 753$	$202k \pm 1.4k$	$370k \pm 2k$

TABLE III
WALL TIME (MS) BY NUMBER OF THREADS

Id	1	6	12	24
1	27.4 ± 0.6	4 ± 0.17	2 ± 0	2 ± 0.4
2	150 ± 1.3	25 ± 0.1	12 ± 0.6	10 ± 0.3
3	164 ± 3	27 ± 0.5	13 ± 0.5	11 ± 0.2
4	$2k \pm 65$	717 ± 19	374 ± 11	224 ± 10
5	$144k \pm 647$	$25k \pm 270$	$14k \pm 483$	$13k \pm 133$
6	$420k \pm 873$	$70k \pm 890$	$43k \pm 103$	$38k \pm 230$
7	$126k \pm 753$	$73k \pm 359$	$56k \pm 432$	$38k \pm 200$

D. Impact of adjusted neighbours

We explore values of adjusted k neighbours from 0, i.e. using only the self-loop-excluded degree, to 1000. Higher values of k increase the hash compute time but might reduce the number of collisions. Graphs with similar high-degree nodes, like Wikidata, benefit the most from a high k. Otherwise, we observe that using a small number of neighbours (10) generally improves the performance, but higher values such as 100 or 1000 rapidly deteriorate them. The single thread wall times are in Table IV, and values bigger than twelve hours ($43M$ ms) are considered out-of-time (OOT).

E. Impact of degree threshold

In all previous experiments, we considered a threshold of $d_{\text{MIN}} = 100$, which is reasonable for STRING graphs but low for larger graphs. In this subsection, we explore the impact of d_{MIN} on the wall time and the number of INGs.

The wall time required decreases substantially while detecting high-degree INGs. The single thread wall times and the number of INGs are in Tables V and VI, respectively.

F. Discussion of identified ING oddities

We have identified several high-degree INGs (see Table VI). By using AHash, $k = 10$, a single thread, $d_{\text{MIN}} = 1000$, and

TABLE IV
SINGLE THREAD TIME (MS) BY k ADJUSTED NEIGHBOURS

Id	0	10	50	100	1000
1	2 ± 0	2.1 ± 0.3	5 ± 0	8 ± 0.1	27 ± 0.6
2	11 ± 0	15 ± 0.4	24 ± 0.3	37 ± 0.4	150 ± 1.3
3	10 ± 0	13 ± 0.4	23 ± 0.3	36 ± 0	164 ± 2.7
4	102 ± 1.4	53 ± 1	202 ± 3	439 ± 6	$2k \pm 65$
5	$10M \pm 247k$	$5k \pm 65$	$25k \pm 244$	$54k \pm 1k$	$144k \pm 1k$
6	OOT	OOT	OOT	OOT	$420k \pm 1k$
7	OOT	$12k \pm 63$	$30k \pm 154$	$57k \pm 1k$	$126k \pm 1k$

TABLE V
SINGLE THREAD TIME (MS) BY DEGREE THRESHOLD d_{MIN}

Id	100	500	1000	10k	100k
1	27.4 ± 0.6	14 ± 0	1 ± 0.1	0	0
2	150 ± 1.3	112 ± 1	49 ± 0	0	0
3	164 ± 2.7	128 ± 1	66 ± 0	0	0
4	2056 ± 65	$1k \pm 7$	473 ± 3	8.5 ± 0	0
5	$144k \pm 647$	$43k \pm 1k$	$12k \pm 286$	0	0
6	$420k \pm 873$	$51k \pm 33$	$33k \pm 903$	$4.8k \pm 8$	$2.4k \pm 5$
7	$126k \pm 664$	$42k \pm 21$	$22k \pm 10$	$2.7k \pm 2$	$2.3k \pm 2$

TABLE VI
NUMBER OF INGS BY DEGREE THRESHOLD d_{MIN}

Id	100	500	1000	10k	100k
1	0	0	0	0	0
2	0	0	0	0	0
3	8	4	1	0	0
4	907	14	3	0	0
5	58	9	8	0	0
6	20756	6462	3749	483	148
7	91294	10872	4298	0	0

accounting for edge weights and node labels, we identified two isoform proteins in STRING Mus Musculus, namely Rpl7a-ps3 and Rpl7a-ps8 in 4ms. It may be the case that one of the two proteins needs to be removed. Similarly, we identified three INGs in KGCOVID in 6ms, considering both node and edge labels: the first is the triple Watasenia-luciferin, Renilla-luciferin and Cypridina-luciferin, all activity related to bioluminescence in different marine species. The second group is a tuple composed of CHEMBL132268 and CHEMBL388581. It is unclear to the authors why these compounds should be topologically identical. Finally, the isomorphic tuple composed of P0DTC9, a protein of SARS-CoV-2 of 2019, and P59595, the analogous core protein in SARS-CoV-1 from 2002. This is a notable oddity, as topological edge prediction methods cannot distinguish the proteins from the two viruses. A solution to these INGs is to integrate more information into the KG. The INGs from other graphs are not analogously interpretable.

VII. CONCLUSIONS

The paper presents an algorithm to identify attributed INGs, which was evaluated for scalability, how it is affected by the hash function, the number of adjusted k neighbours, and the threshold d_{MIN}. The results show that the algorithm has near-linear scalability with the number of cores and that using a small k improves performance, especially in larger graphs. Additionally, increasing d_{MIN} reduces wall time while allowing the detection of less likely INGs.

We identified ING oddities in real-world graphs, such as isoforms in STRING Mus Musculus and 3 INGs in KGCOVID.

The algorithm can be deployed in QC pipelines and used to improve and monitor the quality of graphs.

VIII. ACKNOWLEDGEMENTS

David Bader is supported in part by NSF Grant 2109988.

REFERENCES

[1] R. C. Read and D. G. Corneil, "The graph isomorphism disease," *Journal of Graph Theory*, vol. 1, no. 4, pp. 339–363, 1977.

[2] D. Szklarczyk, A. L. Gable, K. C. Nastou, D. Lyon, R. Kirsch, S. Pyysalo, N. T. Doncheva, M. Legeay, T. Fang, P. Bork *et al.*, "The STRING database in 2021: customizable protein-protein networks, and functional characterization of user-uploaded gene/measurement sets," *Nucleic Acids Research*, vol. 49, no. D1, pp. D605–D612, 2021.

[3] E. Huaman, E. Kärle, and D. Fensel, "Duplication detection in knowledge graphs: Literature and tools," 2020. [Online]. Available: https://arxiv.org/abs/2004.08257

[4] L. Cappelletti, T. Fontana, E. Casiraghi, V. Ravanmehr, T. J. Callahan, M. P. Joachimiak, C. J. Mungall, P. N. Robinson, J. Reese, and G. Valentini, "GraPE: fast and scalable Graph Processing and Embedding," 2021. [Online]. Available: https://arxiv.org/abs/2110.06196

[5] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[6] H. Bast and T. Hagerup, "Fast parallel space allocation, estimation, and integer sorting," *Information and Computation*, vol. 1, no. 123, pp. 72–110, 1995.

[7] S. Edelkamp and A. Weiß, "Blockquicksort: Avoiding branch mispredictions in quicksort," *Journal of Experimental Algorithmics (JEA)*, vol. 24, pp. 1–22, 2019.

[8] J. T. Reese, D. Unni, T. J. Callahan, L. Cappelletti, V. Ravanmehr, S. Carbon, K. A. Shefchek, B. M. Good, J. P. Balhoff, T. Fontana, H. Blau, N. Matentzoglu, N. L. Harris, M. C. Munoz-Torres, M. A. Haendel, P. N. Robinson, M. P. Joachimiak, and C. J. Mungall, "KG-COVID-19: A framework to produce customized knowledge graphs for COVID-19 response," *Patterns*, vol. 2, no. 1, p. 100155, 2021.

[9] D. Vrandečić and M. Krötzsch, "Wikidata: a free collaborative knowledgebase," *Communications of the ACM*, vol. 57, no. 10, pp. 78–85, 2014.

[10] R. A. Rossi and N. K. Ahmed, "The network data repository with interactive graph analytics and visualization," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, ser. AAAI'15. AAAI Press, 2015, p. 4292–4293.

[11] C. L. Clarke, N. Craswell, and I. Soboroff, "Overview of the TREC 2009 web track," DTIC Document, Tech. Rep., 2009.

[12] Y. Collet, "xxHash: Extremely fast hash algorithm," 2016. [Online]. Available: http://cyan4973.github.io/xxHash/

[13] T. Kaitchuck, "aHash," 2019. [Online]. Available: https://github.com/tkaitchuck/aHash

[14] J.-P. Aumasson and D. J. Bernstein, "SipHash: a fast short-input PRF," in *International Conference on Cryptology in India*. Springer, 2012, pp. 489–508.