Improving efficiency of VF3 and VF3-light algorithms for sparse graphs

Daniil I. Dzenhaliou
Faculty of Applied Mathematics and Computer Science Belarusian State University
Minsk, Republic of Belarus
ddengalev@gmail.com

Vladimir I. Sarvanov
Department of Number Theory and Discrete Mathematics Institute of Mathematics, National Academy of Sciences of Belarus Minsk, Republic of Belarus
sarvanov@im.bas-net.by

***Abstract*—Researchers have made notable progress in improving the way we find isomorphic subgraphs in labeled or unlabeled graphs by focusing on efficiency. One group of algorithms, known as the VF series, has consistently shown its effectiveness, especially when dealing with large sparse graphs. In this paper, we introduce a new method that leverages machine learning capabilities, aiming to improve the performance of VF3 and VF3-light algorithms in solving the specified problem. Also, we propose a new parallelization scheme for VF3 and VF3-light algorithms.**

***Index Terms*—graph, algorithm, isomorphism, subgraph isomorphism, parallelization, machine learning**

I. Introduction

The problem of subgraph isomorphism in labeled or unlabeled graphs involves the identification of all isomorphic induced embeddings of a "small" graph pattern (referred to as H) within a "large" data graph (referred to as G). In this paper, we refer to this task as the subgpraph isomorphism problem. This problem has widespread applications and emerged in the field of bioinformatics, where H represented a target subgraph to be located within a larger biomolecule graph, with atoms (C, H, N, O) corresponding to vertices and bonds as edges. Such isomorphic embeddings provided insights into the properties of these biomolecules, crucial for drug discovery and toxicity assessments.

Subsequently, this problem found application in pattern recognition [1]–[3], particularly in the design of complex chips. In this context, ensuring that a design does not contain physically impossible fragments is critical for chip manufacturing. Here, vertices correspond to basic components like transistors, resistors, capacitors, and edges represent their connections (conductors).

More recently, the problem of labeled graph isomorphism with labeled vertices and edges has become extensively used in social network analysis. It enables searching for various useful subgraphs in social networks, such as friends, interest groups, classmates, and other organized communities. These findings have practical implications, including those related to facilitating targeted advertising campaigns.

While the problem of isomorphism between two graphs is efficiently solvable for planar graphs and graphs with bounded vertex degrees, Laszlo Babai introduced a sub-exponential algorithm for general graphs [4], suggesting that the problem is likely not NP-complete. However, the task of searching for a single isomorphic subgraph is NP-hard, as demonstrated by considering H as a complete graph of order k. Furthermore, the problem of finding all isomorphic subgraphs can produce solution sizes that are not bounded by a polynomial in the input size

The "VF" series [5] algorithms are among the main algorithms for solving subgraph isomorphism problem. For example, the VF2 [6] algorithm is included as a solver in the NetworkX [7] and Boost [8] libraries.

VF series algorithms, like most other algorithms for this problem, are based on backtracking. Along with cutoffs, the key role in such algorithms is played by the ordering of pattern vertices, i.e., the order in which pattern vertices are included in backtracking. Choosing a "good" (ideally optimal) ordering can improve the performance of the algorithm. In this paper, we propose a new approach to selecting a "good" ordering using machine learning techniques.

II. Effective vertex ordering

We consider a data graph $G(V, E)$ and a pattern $H(V', E')$. Suppose we have a certain algorithm like VF3 [9] or VF3-light [10]. Let F be the function that assigns to each permutation the runtime of the algorithm when the pattern vertices are executed in the order specified by the permutation. A set of pairs consisting of a data graph and a pattern is then formed, and the runtime is computed for each pair. The obtained information is used as heuristic assumptions to reduce the runtime.

The algorithm's performance is significantly influenced by the order in which pattern vertices are processed. Thus, the natural question arises: how can we find an efficient vertex ordering in a graph? We need to discover an ordering of pattern vertices determined by the permutation $P = p_1, p_2, ..., p_n$, where $n = |V'|$, that effectively reduces the algorithm's execution time, as determined by the function $F : P \to R$. Effective reduction means that the program's execution time with this ordering is significantly better than the average execution time with a random vertex order.

In the VF3-light algorithm, the problem is solved by placing vertices in a sorted list according to three criteria, in decreasing order of priority as detailed below.

Vertices with a large number of outgoing edges to vertices already selected in the list are placed at the beginning.

If multiple vertices have equal scores according to the first criterion, vertices with the smallest number of vertices in the subgraph that have the same or greater degree and label as the current pattern vertex are placed closer to the beginning.

If multiple vertices meet the first and second criteria, vertices with the highest degree are placed closer to the beginning.

We propose an enhancement of the previously described algorithm. The core contribution is the integration of machine learning techniques (graph representation learning algorithms) with local optimal solution-finding algorithms (genetic and simulated annealing algorithms) to enhance the graph vertex ordering algorithm for speeding up the search for isomorphic subgraphs.

We suggest introducing a weight vector $\sigma = (\sigma_1, ...\sigma_k)$ that determines the influence of each value in the vertex representation vector on its position in the ordering. For the first three elements of the representation vector, we employ criteria from the VF3-light algorithm.

- The number of edges outgoing from the current vertex to vertices already ordered.
- The number of vertices in the data graph that have the same or greater degree and label as the current pattern vertex.
- The degree of a vertex in the pattern graph.

The remaining elements of the representation vector are taken from specialized vertex representations, such as Node2Vec [11], Struc2Vec [12], Role2Vec [13]. These representations should reflect the structure of the vertex and its environment to be suitable for (vertex degree, neighbor degrees, etc.). The utilization of vertex representations already plays a pivotal role in enhancing the efficiency and accuracy of various graph-based machine learning tasks, enabling the extraction of valuable information and patterns from individual nodes within a network. Hence, algorithms like Struc2Vec and Role2Vec are suitable for filling remaining elements of the representation vector.

To generate an optimal weight vector σ, a sufficiently large set of graphs needs to be collected so that the resulting vector can be applied to a wide range of graphs. This dataset should consist of pairs of datagraphs and patterns. Alternatively, the dataset can be specialized to make the weight vector more effective for a specific set of graphs.

In this paper, we have additionally gathered a test dataset to assess the enhancement in performance. The pairs of graphs used in the test dataset are generated from the same source as the training dataset, but the datasets themselves do not overlap.

Therefore, the algorithm for efficient graph vertex ordering can be outlined as follows:

1) Gather a set of graphs on which we will get the weight vector σ.
2) Apply a vertex representation algorithm to each pattern graph in the set, chosen in advance for all graphs in the set, and save the resulting representations.
3) Use a genetic algorithm or simulated annealing algorithm to get the weight vector σ.
4) Utilize the weights obtained in step 3 for efficient ordering of other graphs. Order the vertices of the pattern graph based on the ascending scalar product of the weight vector σ and the vertex representation vector.

Now we turn to a detailed discussion of Step 3 of the algorithm as we propose a variant of Step 3 using a genetic algorithm [14]. This approach involves the following steps.

Generate a "population" (a set of k vectors of size n), where k is the population size, typically ranging from 10 to 100. Perform m iterations of the algorithm (executing steps 3.1) - 3.3) is considered one iteration).

3.1) Take a random pair of vectors from the population and perform "crossover" - the result of the crossover for two vectors σ_i and σ_j will be σ = (σ_i + σ_j) / 2.
3.2) Apply mutation to the vectors obtained in step 3.1) - add a randomly generated vector σ_{rnd} independently to each vector, with the constraint $\sigma_{rnd} \leq \epsilon$, where ϵ is a small constant.
3.3) Add the resulting vectors to the population, calculate the execution time value on the collected sample, and keep only the top k vectors (with the smallest average execution time). This marks the completion of one iteration.

Note: Calculating the average execution time on all graphs in this algorithm may lead to incorrect results. Instead, it is suggested to use the natural logarithm of the execution time.

Next, we can provide a variant of step 3 for the simulated annealing algorithm [14]. We define $F(\omega) = \sum_{G \in S} \log(time(\omega, G))$, where S is the set of all vectors from the sample collected in step 1, and $time(\omega, G)$ is the execution time of the algorithm on graph G when using the weight vector ω.

3.1) Generate a random vector ω as the initial state. Set an initial temperature T.
3.2) Generate a random vector ω_{rnd}, where $|\omega_{rnd}| \leq \epsilon$, and ϵ is a small constant. Check the following condition: if $F(\omega + \omega_{rnd}) \leq F(\omega)$, then replace ω with $\omega + \omega_{rnd}$. Otherwise, replace it with a probability of

$$\exp\left(\frac{-(F(\omega + \omega_{rnd}) - F(\omega))}{T}\right) \quad (1)$$

Multiply T by a constant $\sigma < 1$.
3.3) If $T \leq \epsilon'$, where ϵ' is a pre-defined threshold, terminate the algorithm; otherwise, repeat step 3.2)

Finally, we present a version of Step 3 designed for the random search algorithm:

Run the algorithm k times.

In each run of the algorithm, do the following:

3.1) Generate q random weight vectors ω and choose the best one (the one where the function F takes the minimum value).
3.2) Perform m iterations of the algorithm. On each iteration, generate a random vector ω_{rnd} as follows: choose a random number s from 0 to n and set non-zero values

at s positions, and set zero values at the rest. Replace ω with $\omega + \omega_{rnd}$ only when $F(\omega) > F(\omega + \omega_{rnd})$.

3.3) Choose the best weight vector ω among all values obtained in the algorithm runs. This obtained value is locally optimal after a sufficient number of algorithm iterations.

The version employing random search is used for the experiments in this study due to its faster convergence rate in practical experiments.

It should be noted that the algorithm demonstrates optimal performance when the set of graphs collected for weight vector tuning and the set of graphs for algorithmic application share the same pattern graph. This similarity implicitly defines an efficient ordering for the pattern graph, making the algorithm useful even when paired with a different data graph. However, the experiments conducted in this study employed varying pattern graphs, suggesting that the algorithm also performs efficiently under these conditions.

III. Pattern decomposition

We consider a data graph $G(V, E)$ and a pattern graph $H(V', E')$.

A partition of a graph X, denoted as $X_1, ..., X_k$, is defined as follows:

$VX_1 \cup VX_2, ..., \cup VX_k = VX$, for all i, j, where $i \neq j$, $VX_i \cap VX_j = \emptyset$. Additionally, $X_1, ..., X_k$ are induced subgraphs of graph X.

The decomposition algorithm works as follows:

1) Assume we have an arbitrary partition of the graph H as $H_1, ..., H_k$.
2) Find all isomorphic induced occurrences of graphs $H_1, ...H_k$ within graph G. Denote the corresponding subgraphs in data graph G for graph H_i as $L_{i,1}, ..., L_{i,n_i}$.
3) Define graphs Q_i as follows:

$$Q_1 = L_{1,1}, ..., Q_{n_1} = L_{1,n_1}, Q_{n_1+1} = L_{2,1}, ..., Q_p = L_{k,n_k} \quad (2)$$

where $p = \sum_{i=1}^{k} n_i$.
4) Construct a new graph G' as follows: Let $V = 1, 2, ...p$, add an edge incident to vertices i and j, $i \neq j$, if and only if Q_i and Q_j correspond to different graphs H_x and H_y, do not share any vertices, and between the vertices of Q_i and Q_j in G, the same edges are present as between the vertices of H_x and H_y in H.
5) Find all cliques of size k in the obtained graph. If a clique contains vertices $(i_1, ..., i_k)$, construct $(x_1, ...x_k)$ such that Q_{i_j} corresponds to H_{x_j}. Then, vertices from Q_{i_1} in the data graph correspond to vertices of H_{x_1}, vertices from Q_{i_2} correspond to vertices of H_{x_2}, and so on. So, each clique represents one subgraph isomorphism, with Q_{i_j} as the embeddings in the datagraph and H_{x_j} as the pattern.

We can now prove that this approach enables us to find all isomorphic embeddings of H in G.

First, we aim to demonstrate that step 4 yields isomorphic embeddings.

Since all H_{x_j} are distinct according to step 4, this means that the union of all H_{x_j} forms the graph H. All Q_{i_j} are induced embeddings of H_{x_j}, so all edges in Q_{i_j} will correspond to the edges in H_{x_j}. The vertices of Q_{i_j} will also correspond to the vertices of H_{x_j}. The set obtained by combining all vertices from H_{x_j} is equal to VH, and all vertices from $Q_{i_{j_1}}$ and $Q_{i_{j_2}}$ are distinct for any $j_1 \neq j_2$. The edges between $Q_{i_{j_1}}$ and $Q_{i_{j_2}}$, where $j_1 \neq j_2$, will be the same as between the components $H_{x_{j_1}}$ and $H_{x_{j_2}}$. There are no other edges in the graph, so the union of Q_{i_j} indeed forms an isomorphic embedding.

Now, we prove that we enumerate all isomorphic embeddings. We assume that some embedding is not found using this algorithm. We find the vertices corresponding to H_i in graph G. For each i, they form an induced subgraph Q_{x_i}. These graphs have labels x_i in the graph G' obtained in step 4 of the algorithm. Graphs Q_i and Q_j, where $i \neq j$, $i, j \in x_1, ..., x_k$, correspond to different subgraphs in the partition of H, do not share vertices, and the edges between vertices of Q_i and Q_j in G' are the same as between vertices of H_i and H_j in H. Therefore, in the graph obtained in step 4 of the algorithm, the vertices $x_1, ..., x_k$ form a clique. This leads to a contradiction.

Since the verification of the correctness of partial mapping in one iteration takes $O(N^2)$ time, where N is the number of vertices in the pattern, in the pattern decomposition algorithm, this check is done in $O(\frac{N^2}{k})$ time. The additional cost in the algorithm is the time spent on finding all cliques of size k. However, in practice, the algorithm is well-suited for parallelization since isomorphic embeddings need to be found independently for all subgraphs in the partition.

IV. Experiments

In our experiments, we utilized the Attributed Relational Graph (ARG) Database [16], a comprehensive repository of attributed relational graphs that represent various domains or datasets. These graphs have been meticulously designed to capture complex relationships and attributes associated with nodes and edges. This makes them particularly suitable for a diverse range of research applications, including our own. The database encompasses a wide spectrum of domains, including but not limited to:

- Social Networks
- Biological Networks
- Transportation Networks
- Citation Networks

In addition to graph structure, the ARG Database includes labels associated with both nodes and edges. This labeled data allows for more nuanced and context-aware analyses, making it particularly valuable for understanding complex relationships in real-world data.

The graphs within the ARG Database often mirror real-world complexities. This characteristic makes them well-suited for exploring the challenges posed by real-world scenarios, enabling researchers to develop solutions that address practical issues.

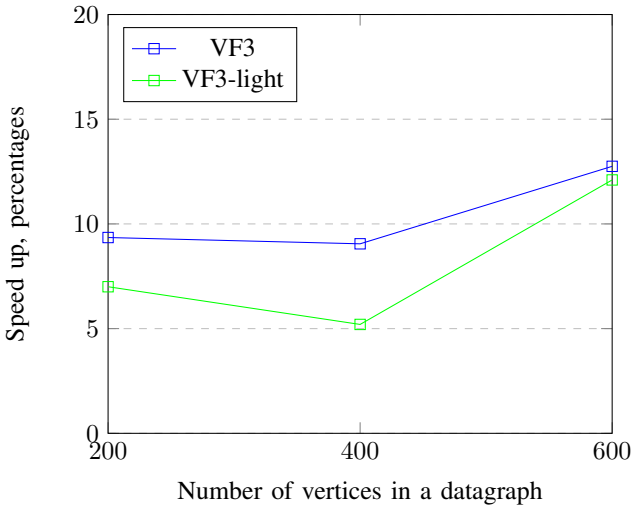

Fig. 1: VF3 and VF3-light exectution speedup using the proposed effective vertex ordering algorithm

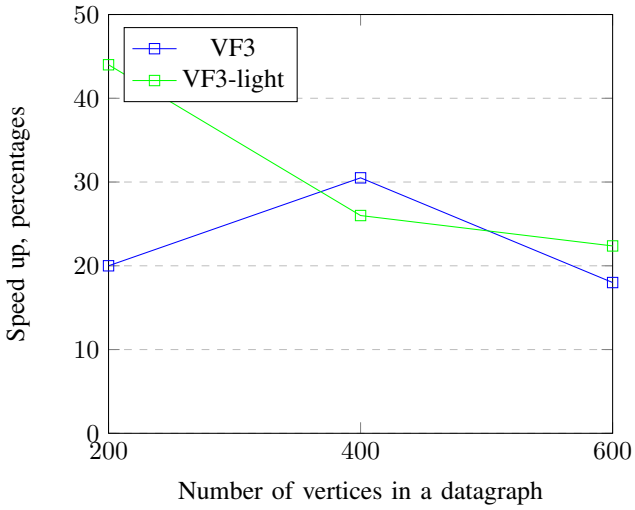

Fig. 2: VF3 and VF3-light execution speedup using pattern decomposition

ARG Database consists of many different graph types. However, proposed algorithm improvements significantly improve efficiency of VF3 and VF3-light algorithms only on specific types.

Effective ordering improves VF3 and VF3-light performance on bounded-valence graphs with valence = 3. At the first plot (Fig. 1) VF3 and VF3-light execution speedup dependence on size of a datagaraph is shown, size of a pattern = 20% of the datagraph size.

Pattern decomposition also improves VF3 and VF3-light performance on bounded-valence graphs with valence = 3. So, the second plot shows the execution speedup improvement. Pattern decomposition works efficiently when the size of a pattern is enough big (bigger than 50% of the datagraph size)

At the second plot (Fig. 2) VF3 and VF3-light execution speedup dependence on size of a datagaraph is presented, size of a pattern = 60% of the datagraph size.

V. Conclusion

The main results and primary contributions of this paper are as follows.

1) Modifications of the VF3 and VF3-light algorithms were developed based on pattern decomposition, enabling efficient parallelization of computations. This parallelization can be used for solving subgraph isomorphism problems for large sparse graphs.
2) A new algorithm for solving the efficient node ordering problem in a pattern graph was developed and implemented. This algorithm utilizes machine learning methods, representing a novel approach to solving this problem. The proposed algorithm is used to improve VF3 and VF3-light performance and, possibly, to improve other algorithms that solve the subgragh isomorphism problem.
3) New modifications of the VF3 and VF3-light algorithms were proposed, incorporating efficient node ordering for the pattern graph. These modifications improve VF3 and VF3-light performance on sparse graphs.
4) Conditions for the preferred use of each modification were determined based on the analysis of experimental results. These conditions can be used to understand where proposed modifications are applicable. This analysis revealed that:
   - The performance of the modified algorithm using decomposition is better than the performance of standard versions of VF3 and VF3-light for bounded-valence with valence=3 and large enough query graphs.
   - Modified algorithms using optimized node ordering perform faster than the standard versions of VF3 and VF3-light on bounded-valence with valence=3. It is worth noting that VF3 and VF3-light are considered the most suitable for application to sparse graphs

References

[1] P. Foggia, G. Percannella, and M. Vento, “Graph Matching and Learning in Pattern Recognition in the Last 10 Years,” International Journal of Pattern Recognition and Artificial Intelligence, Feb. 2014, doi: 10.1142/S0218001414500013.

[2] M. Vento, “A Long Trip in the Charming World of Graphs for Pattern Recognition,” Pattern Recogn., vol. 48, no. 2, pp. 291–301, Feb. 2015, doi: 10.1016/j.patcog.2014.01.002.

[3] D. Conte, P. Foggia, C. Sansone, and M. Vento, “Thirty Years Of Graph Matching In Pattern Recognition,” IJPRAI, vol. 18, pp. 265–298, May 2004, doi: 10.1142/S0218001404003228.

[4] L. Babai, Graph Isomorphism in Quasipolynomial Time. 2015.

[5] V. Carletti, P. Foggia, A. Saggese and M. Vento, ”Challenging the Time Complexity of Exact Subgraph Isomorphism for Huge and Dense Graphs with VF3,” in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 40, no. 4, pp. 804-818, 1 April 2018, doi: 10.1109/TPAMI.2017.2696940.

[6] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (Sub)Graph Isomorphism Algorithm for Matching Large Graphs," IEEE Trans. Pattern Anal. Mach. Intell., vol. 26, no. 10, pp. 1367–1372, Oct. 2004, doi: 10.1109/TPAMI.2004.75.
[7] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring Network Structure, Dynamics, and Function using NetworkX," in Proceedings of the 7th Python in Science Conference, 2008, pp. 11–15.
[8] Boost, Boost C++ Libraries. http://www.boost.org/, 2023. [Online]. Available: http://www.boost.org/
[9] C. Vincenzo, P. Foggia, A. Saggese, and M. Vento, "Introducing VF3: A New Algorithm for Subgraph Isomorphism," May 2017, pp. 128–139. doi: 10.1007/978-3-319-58961-9_12.
[10] C. Vincenzo, P. Foggia, A. Greco, M. Vento, and V. Vigilante, "VF3-Light: a lightweight Subgraph Isomorphism Algorithm and its Experimental Evaluation," Pattern Recognition Letters, vol. 125, Jul. 2019, doi: 10.1016/j.patrec.2019.07.001.
[11] A. Grover and J. Leskovec, "Node2vec: Scalable Feature Learning for Networks," in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 855–864. doi: 10.1145/2939672.2939754.
[12] L. F. R. Ribeiro, P. H. P. Saverese, and D. R. Figueiredo, "Struc2vec: Learning Node Representations from Structural Identity," in Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2017, pp. 385–394. doi: 10.1145/3097983.3098061.
[13] N. Ahmed et al., "Learning Role-based Graph Embeddings," Feb. 2018.
[14] K. Sastry, D. Goldberg, and G. Kendall, "Genetic Algorithms," in Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, E. K. Burke and G. Kendall, Eds. Boston, MA: Springer US, 2005, pp. 97–125. doi: 10.1007/0-387-28356-0_4.
[15] E. H. L. Aarts, J. H. M. Korst, and P. J. M. Laarhoven, van, "Simulated annealing," in Local search in combinatorial optimization, E. H. L. Aarts and J. K. Lenstra, Eds. Wiley-Interscience, 1997, pp. 91–120.
[16] M. De Santo, P. Foggia, C. Sansone, and M. Vento, "A large database of graphs and its use for benchmarking graph isomorphism algorithms," Pattern Recogn. Lett., vol. 24, no. 8, pp. 1067–1079, May 2003, doi: 10.1016/S0167-8655(02)00253-2.