Simple Pattern-only Heuristics Lead To Fast Subgraph Matching Strategies on Very Large Networks.

Antonino Aparo, Vincenzo Bonnici, Giovanni Micale, Alfredo Ferro, Dennis Shasha, Alfredo Pulvirenti, Rosalba Giugno*

University of Verona, Verona, Italy; University of Catania, Catania, Italy;
New York University, New York, USA
rosalba.giugno@univr.it

Abstract. A wide range of biomedical applications entails solving the subgraph isomorphism problem, i.e. finding all the possible subgraphs of a target graph that are structurally equivalent to an input pattern graph. Targets may be very large and complex structures compared to patterns. Methods that address this NP-complete problem use heuristics. Their performance in both time and quality depends on a few subtleties of those heuristics. This paper compares the performance of state-of-the-art algorithms for subgraph isomorphism on small, medium and very large graphs. Results show that heuristics based on pattern graphs alone prove to be the most efficient, an unexpected result.

Keywords: Subgraph isomorphism, Networks biology, Search strategy

1 Introduction

In the last decade, significant national, international and private research resources have been directed at data-driven genome-level projects, such as the 1000 Genomes Project[1], Encyclopedia of DNA Elements Project [2], and The Cancer Genome Atlas Project (TGCA)[3]. Different approaches have been proposed to integrate data across multiple omics (i.e. whole species) data sets [11]. Well-known examples are protein-protein interaction networks, genetic regulatory networks, and metabolic networks. Analysis of such networks has lead to the discovery of recurrent and statistically over-represented sub-networks[8, 10].

Such biomedical applications entail solving the subgraph isomorphism problem, an NP-complete problem [9]. The problem consists of finding all the possible subgraphs of a reference graph (called target) that are structurally equivalent to another graph (called the pattern)[2]. Given the potentially exponential problem complexity and the ever growing target graphs, heuristics for subgraph isomorphism algorithms must achieve scalability for large networks [5, 1].

[1] http://www.internationalgenome.org/
[2] http://www.encodeproject.org
[3] https://cancergenome.nih.gov/

We have focused on the two most scalable graph searching algorithms, RI [3, 2] and VF3 [4]. In the literature, such algorithms have been compared on graphs up to 10000 vertices[4]. In the present paper, we have generated a benchmark of graphs containing 1008 target graphs of size up to 20.000 vertices using three different graph generation models (Erdös-Rényi, Barabási, Forest Fire), and about 150.000 pattern graphs of different sizes and densities. Our results show that heuristic strategies based only on patterns perform well on small graphs and perform considerable better on large graphs w.r.t strategies which account also the target graphs.

2 An annotated overview of subgraph isomorphism algorithms

Basic notions and problem definition. A graph G is a pair (V, E), where V is the set of vertices and $E \subset V \times V$ is the set of edges connecting these vertices. G is *labeled* when a set of labels A is assigned to the vertices and/or the edges of G using an injective function: a vertex label function $\alpha : V \to A$ and/or an edge label function $\beta : E \to A$. Labeled graphs can be represented by a quadruple (V, E, α, β). A graph is *directed* if each edge has a direction from the source vertex to the destination vertex, otherwise the graph is *undirected*. *Dense graphs* are those for which the ratio $|E|$ / $|V|$ is relatively high (e.g. the number of edges approaches the square of the number of vertices), where $|V|$ and $|E|$ are the cardinalities of the two sets. Otherwise, graphs are considered *sparse*. Given a *pattern graph* $G_p = (V_p, E_p, \alpha_p, \beta_p)$ and a *target graph* $G_t = (V_t, E_t, \alpha_t, \beta_t)$ the subgraph isomorphism problem is to find a injective function $f : V_p \to V_t$, mapping each vertex of G_p to a unique vertex of G_t, while satisfying the following conditions: (i) $\forall v \in V_p f(v) \in V_t$ (ii) $\forall u, v \in V_p : u \neq v \Rightarrow f(u) \neq f(v)$; (iii) $\forall (u, v) \in E_p \Rightarrow \exists (f(u), f(v)) \in E_t$; (iv) $\forall v \in V_p \Rightarrow \alpha_p(v) = \alpha_t(f(v))$ *and* $\forall (u, v) \in E_p \Rightarrow \beta_p(u, v) = \beta_t(f(u), f(v))$. This is an injective mapping because the target graph may have edges that the pattern graph doesn't. As mentioned in the introduction, the subgraph isomorphism problem is NP-complete [9].

Features of subgraph isomorphism algorithms. Heuristic algorithms to solve the subgraph isomorphism problem build on two main paradigms: tree search ([3, 4]) and constraint programming ([7, 13, 12]). In the *Tree Search* approach, for each pattern graph, a tree is formed containing all possible mappings of the pattern into the target graph. A path from the dummy root to a leaf contains $|V_p|$ nodes (i.e. the number of nodes in the pattern), each node in the tree is a mapping $(v, f(v))$ with $v \in V_p$. A path from the root to an internal node represents a partial matching, because not all vertices in the pattern graph have been mapped up to that point. The tree is obtained by incrementally extending a partial solution with a new pair of vertices $(v, f(v))$ that satisfies the subgraph isomorphism constraints. If there are no more candidate pairs, the algorithm backtracks and prunes that branch. An advantage of this approach is that the

algorithms have linear memory complexity with respect to the number of vertices.

The *constraint programming* approach is based on the pre-computation of compatibility domains, i.e. pattern vertices that are associated to sets of target vertices that satisfy subgraph isomorphism constraints. These pre-computed domains are used during the matching phase to select candidate pairs. At each matching step, domains are reduced by propagating the constraints from the matched vertices to the unmatched ones and filtering away those vertices that can not be contained in a final solution. The main problem with this approach is that it consumes a lot of memory, making it not scalable on large graphs.

In summary, Single Search Tree (SST) approaches offer better performance and scalability than constraint programming based approaches[2]. The most important heuristic choices of SST-based algorithms are the variable ordering and how the tree is visited.

Variable ordering, also known as the search strategy, is the determination of an ordering of the pattern graph vertices in the branches of the search tree. The order can be *static* or *dynamic*. Static ordering means that the ordering is fixed a priori, before the search phase starts, and remains the same for all paths through the tree. In dynamic ordering, the ordering can change for different branches, implying additional computation during the traversal of the tree. Search strategies differ depending on which properties and which graphs are examined. An algorithm can consider features only of the target graph (*target-dependent*) [6] or only of the pattern graph (*pattern-dependent*) [3] or both [4]. State-driven variable ordering [3, 4] is based on the current state of computation, while domain-driven [6] regards the values in the variable domains .

The best-performing (and most recent) algorithms using an SST approach are VF3 [4] and RI [2, 3].

VF3 adopts a *dynamic*, *target-dependent* and *pattern-dependent* variable ordering. A scoring function gives a priority to the vertices that have more constraints to satisfy, taking into consideration also the probability of finding a compatible vertex in the target graph. The algorithm in the preprocessing phase partially pre-computes sets, called *feasibility sets*, for each vertex. Given a state s of the SST, during the matching process, classification functions are used to subdivide the feasibility sets, so that if two nodes belong to different classes they will never be coupled together. VF3 uses *look-ahead* procedures to predict downward inconsistency, applying constraints and properties related to the current partial matching to the neighbors of a vertex.

RI uses a *static* variable ordering and forms the tree in a *target-independent* way. Static variable ordering strategy allows RI to reduce the search space without using predictive pruning verification (which can be expensive). The vertices are sorted by a score function that takes into account the neighbors of a vertex in the pattern graph alone. The goal of the ordering is to start with the parts of the pattern graph that must satisfy the most possible constraints, so can be filtered away most quickly. During the matching phase, the vertices of the pattern graph are compared with those of the target graph using the search tree.

As the search tree is traversed the subgraph isomorphism conditions are tested. If they are not satisfied, then RI *backtracks* (cutting the corresponding branch and reducing the search space).

3 Results

In order to compare RI and VF3 across a variety of scenarios, we have generated a dataset containing graphs obtained according to stochastic (*Erdös-Rényi*) and scale-free models (*Barabási* and *Forest-Fire*), starting from small (100 node) and sparse graphs up to large (20,000 node) and dense graphs both with few or many labels. We used the python library *network* (for Erdös-Rényi graphs) and *igraph* (for Bababasi and Forest-Fire graphs). Altogether, our benchmark consists of 1,008 target graphs.

After the generation of target graphs, a *random walk based algorithm* was used to extract the patterns by varying pattern size and density. Starting from a vertex of the target graph, a neighbor is selected randomly, until the desired number of vertices of the pattern is reached. Subsequently the algorithm adds to the pattern the edges not yet selected among the chosen vertices, until the specified density is reached.

Tests were running on a machine equipped with an Intel Core i7-7700 3.60GHz 8-core CPU and 15 GB of RAM running a Linux Ubuntu 17.04 64bit operating system. A 3 minute timeout has been set for both algorithms.

Comparisons on the Erdös-Rényi dataset. A graph G with parameter p is created by connecting its N vertices randomly. Each edge is included in the graph with probability p independently of any other edge. $E = pN(N-1)$ is the expected number of edges. This dataset contains 464 targets graphs of 100, 200, 500, 1,000, 2,000, 5,000, 10,0000, 20,0000 vertices with labels ranging from 0.1% to 30% depending on the number of vertices and with probability p: 0.001, 0.002, 0.005, 0.01, 0.02, 0.05, 0.1, 0.2, 0.3, 0.4. From these targets, up to 240 pattern graphs were extracted for each target with 4, 8, 16, 24, 32, 64, 128, 256 number of vertices and three different pattern densities: 0.1 (sparse), 0.5 (medium dense), and $\approx$1 (dense). Figure 1 reports running times of the two algorithms on the *Erdös-Rényi* dataset. Times are grouped by the number of target vertices of the matching instances (Figure 1(a)). The majority of the RI times are below the VF3 boxes and few outliers have comparable execution times with VF3. The other three plots show running times on target graphs having 10k vertices. Both algorithms increase the time requirement by incrementing the number of target vertices and the value of the density parameter p. RI shows a lower dependence to these factors. For example, for the maximum value of p, VF3 triples its average running time, while the time of RI is less affected. Times for $p > 0.2$ were not reported since for graphs having 10k or 20k nodes, VF3 went in timeout the 100% of cases whereas RI only the 35%. The dependence of both algorithms on the percentage of target labels is shown in Figure 1(c). The variation in execution time of RI is less than that of VF3, and average times are faster using RI. In general, as expected, the greater the number of distinct labels

the less the time required, because fewer target vertices are compatible with the pattern vertices.

Comparisons on the Barabási dataset. Genetic networks may show complex topology, containing few *hubs*, i.e. vertices with a high number of interactions compared to the rest of the network vertices. For example, in a metabolic network, hubs are molecules like ATP or ADP, energy carriers involved in a large number of chemical reactions. A model based on these characteristics reproduces the observed stationary *scale-free distributions* similar to the *Barabási-Albert* one. The network starts with m_0 vertices. At each time step a new vertex is added to the network whit $m(\leq m_0)$ links that connect it to existing vertices in the network. The probability p_i that a link of the new vertex connects to an existing vertex i is proportional to the degree k_i of i $p_i = \frac{k_i}{\sum_j k_j}$; where the sum is made over all existing vertices j. After t time steps, the Barabási-Albert model generates a network with $N = t + m_0$ vertices and $m_0 + m_t$ links.

This dataset has 384 target graphs having 200, 500, 1,000, 5,000, 10,000, 20,000 vertices, different numbers of outgoing edges (1, 2, 3, 6) and different numbers of labels in relation to the number of vertices (0.1%, 1%, 10%). 28,800 patterns were extracted by varying the number of vertices (4, 8, 24, 32, 64) and density (0.1, 0.5, ≈1). Results are shown in figure 2. Times are grouped as in the Figure 1. Running times are faster than in the previous dataset, thus in Figure 2(a) we report times in log scale. RI shows a general behavior faster than the competitor and less dependence from the target features variability.

Due to space constraints, we don't report and picture details on the Forest-Fire model datasets. Briefly, the obtained results for the Forest-Fire model are similar to those for the Barabási-Alber model, but, both algorithms have a few more outliers. Figure 3 shows that RI generally outperforms VF3 on small, medium and very large graphs. Charts show the number of times the tested algorithms are faster than the competitor. They are built by taking into account every combination of target and pattern graph on the three models, and the amounts are expressed as a percentage of wins on the total tests. VF3 reaches its maximum of wins the 8.6% instances on the Barabási-Alber model with 1k target vertices.

4 Conclusion

Fast algorithms for subgraph isomorphism depend on a variety of heuristics, some of which have become quite sophisticated. This paper has shown that a simple approach that depends only on the pattern graph such as RI works better especially for large, dense target graphs where the running time of the matching becomes larger. This is of course a field of ongoing research.

References

1. Bonnici, V., Busato, F., Micale, G., Bombieri, N., Pulvirenti, A., Giugno, R.: Appagato: An approximate parallel and stochastic graph querying tool for biological

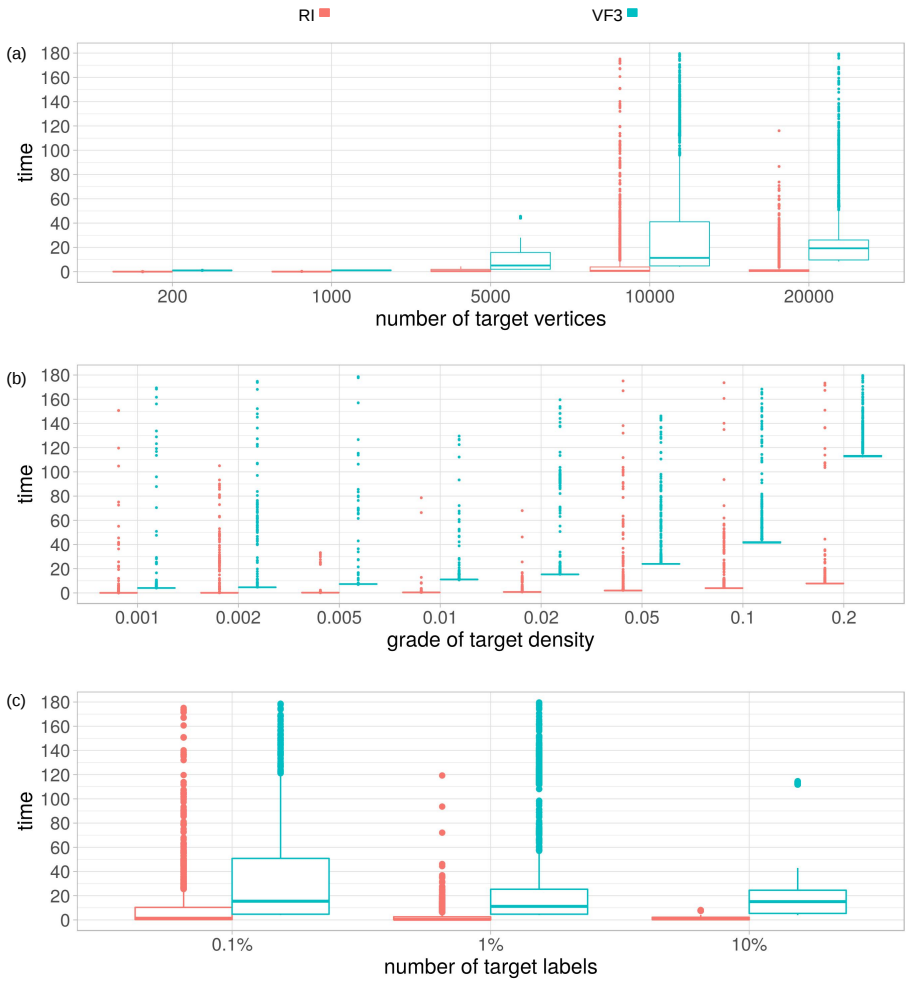

Fig. 1. Comparison of RI (red) and VF3 (green) on the Erdös-Rényi dataset. Execution time is expressed in seconds. (a) Scalability of the algorithms on the size of the target graph, the x-axis reports the number of vertices of the graph. (b) Scalability of the algorithms as a function of the density of the target graph having 10k vertices. Here, the x-axis the probability p of the Erdös-Rényi models. (c) Performance of the two algorithms on the number of labels in the target graphs having 10k vertices, the labels express the percentage with respect to the number of vertices.

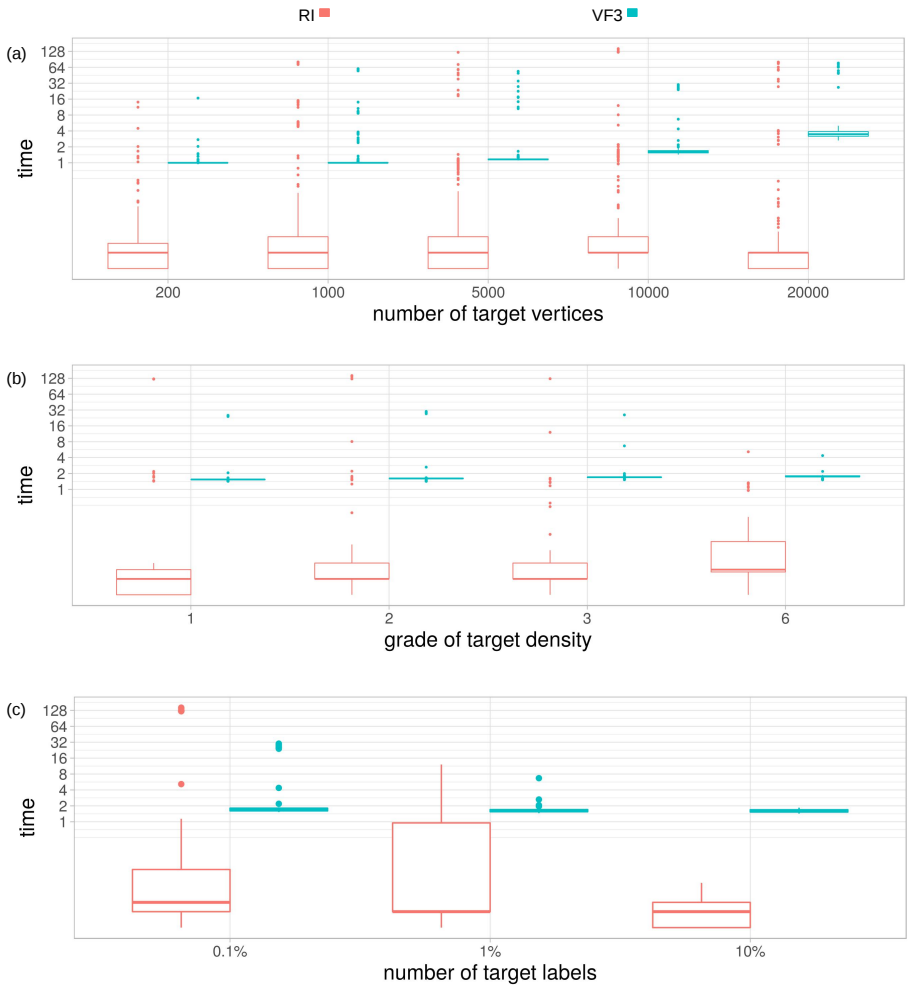

Fig. 2. Comparison of RI (red) and VF3 (green) on the *Barabási-Albetr* dataset. Execution time is expressed in seconds. (a) Scalability of the algorithms on the size of the target graph, the x-axis refers to the number of vertices of the graph. Times are expressed in log scale. (b) Scalability of the algorithms on the density of the target graph having 10k vertices, the x-axis the parameter m of the Barabási-Albert models. (c) Performance of compared algorithms on the number of labels in the target graphs having 10k vertices, the labels express the percentage w.r.t. the number of vertices.

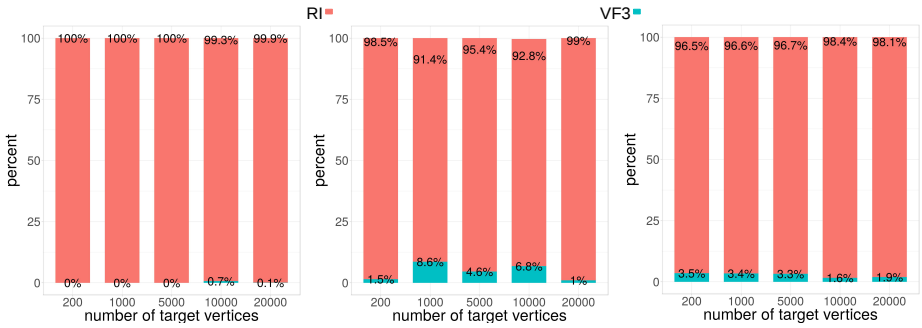

Fig. 3. Number of times, expressed as percentages w.r.t. the total amount of tests, that one algorithm (RI is red and VF3 is green) has been faster than the other. The results are grouped by number of target vertices for the Erdös-Rényi, Barabási-Albert and Forest Fire models, respectively from the left to the right.

networks. Bioinformatics **32**(14) (2016) 2159–2166 Cited By :2.
2. Bonnici, V., Giugno, R.: On the variable ordering in subgraph isomorphism algorithms. IEEE/ACM Transactions on Computational Biology and Bioinformatics **14**(1) (Jan 2017) 193–203
3. Bonnici, V., Giugno, R., Pulvirenti, A., Shasha, D., Ferro, A.: A subgraph isomorphism algorithm and its application to biochemical data. BMC Bioinformatics **14 Suppl 7** (2013) S13
4. Carletti, V., Foggia, P., Saggese, A., Vento, M.: Challenging the time complexity of exact subgraph isomorphism for huge and dense graphs with vf3. IEEE Transactions on Pattern Analysis and Machine Intelligence **PP**(99) (2017) 1–1
5. Giugno, R., Bonnici, V., Bombieri, N., Pulvirenti, A., Ferro, A., Shasha, D.: Grapes: A software for parallel searching on biological graphs targeting multi-core architectures. PLoS ONE **8**(10) (2013) Cited By :10.
6. Haralick, R.M., Elliott, G.L.: Increasing tree search efficiency for constraint satisfaction problems. Artificial Intelligence **14**(3) (1980) 263 – 313
7. McGregor, J.: Relational consistency algorithms and their application in finding subgraph and graph isomorphisms. Information Sciences **19**(3) (1979) 229 – 250
8. Micale, G., Giugno, R., Ferro, A., Mongiovì, M., Shasha, D., Pulvirenti, A.: Fast analytical methods for finding significant labeled graph motifs. Data Mining and Knowledge Discovery (Nov 2017)
9. Michael, R.G., David, S.J.: Computers and intractability: a guide to the theory of np-completeness. WH Free. Co., San Fr (1979) 90–91
10. Milo, R., Shen-Orr, S., Itzkovitz, S., Kashtan, N., Chklovskii, D., Alon, U.: Network motifs: Simple building blocks of complex networks. Science **298**(5594) (2002) 824–827
11. Palsson, B., Zengler, K.: The challenges of integrating multi-omic data sets. Nature Chemical Biology **6** (Oct 2010) 787 EP –
12. Solnon, C.: Alldifferent-based filtering for subgraph isomorphism. Artificial Intelligence **174**(12) (2010) 850 – 864
13. Ullmann, J.R.: Bit-vector algorithms for binary constraint satisfaction and subgraph isomorphism. J. Exp. Algorithmics **15** (February 2011) 1.6:1.1–1.6:1.64