Pattern Recognition Letters 125 (2019) 591–596

Contents lists available at ScienceDirect

Pattern Recognition Letters

journal homepage: www.elsevier.com/locate/patrec

VF3-Light: A lightweight subgraph isomorphism algorithm and its experimental evaluation

Vincenzo Carletti*, Pasquale Foggia, Antonio Greco, Mario Vento, Vincenzo Vigilante

Dep. of Information and Electrical Engineering and Appl. Mathematics, University of Salerno, Italy

ARTICLE INFO

Article history:
Received 17 February 2019
Revised 16 May 2019
Accepted 1 July 2019
Available online 2 July 2019

MSC:
68R10

Keywords:
Graph matching
Subgraph isomorphism
Structural pattern recognition

ABSTRACT

In this paper we introduce VF3-Light, a simplification of VF3, a recently introduced, general-purpose subgraph isomorphism algorithm. While VF3 has demonstrated to be very effective on several datasets, especially on very large and very dense graphs, we will show that on some classes of graphs, the full power of VF3 may become an overkill; indeed, by removing some of the heuristics used in it, and as a consequence also some of the data structures that are required by them, we obtain an algorithm (VF3-Light) that is actually faster.

In order to provide a characterization of this modified algorithm, we have performed an evaluation using several publicly available graph datasets. Besides comparing VF3-Light with VF3, we have also included in the comparison other recent algorithms that are rated among the fastest in the state of the art.

1. Introduction

Graph-based representations are widely adopted in *Structural Pattern Recognition*, where the objects of interest can be decomposed into parts (represented as *nodes*) and significant information is attached to the relationships between parts (represented as *edges*). This kind of representation has been profitably used in applications such as computer vision, chemistry, biology, social network analysis, databases.

A task that is common to several applications is the search for a correspondance between the structures of two graphs (*graph matching*); an important special case is the search for occurrences of a smaller *pattern graph* inside a larger *target graph*. *Subgraph isomorphism* is a possible formulation of this problem, widely investigated in the literature: see [9,11,15] for extensive reviews on subgraph isomorphism and other graph matching algorithms in the field of Pattern Recognition.

Many subgraph isomorphism algorithms (e.g. [14], VF2 by [10], L2G by [1], RI/RI-DS by [2]) are based on *Tree Search*. In this approach, a *search space* (also called *state space*) is conceptually defined as a tree of *states*, where each state correspond to a partial mapping of the pattern nodes onto target nodes. The root of the tree is the state corresponding to an empty mapping, while a new state is obtained from an existing one by adding to the mapping a pair (pattern node, target node) that ensures the preservation of the structural constraints imposed by problem formulation. Algorithms based on this approach perform a depth-first visit of the state space with backtracking, in order to avoid keeping in memory the whole search tree. The algorithms essentially differ from each other in the order they visit the search space, in the heuristics they adopt for pruning unfruitful portions of the space, and in the data structures they need to keep and update during the visit process. These differences, although they do not change the asymptotic worst-case complexity (the problem is NP-complete), may greatly affect the actual execution times on graphs commonly found in applications.

The definition of the heuristics is often the most critical part in the design of the algorithm: on one hand, a very accurate heuristic may allow the algorithm to detect in advance that a candidate state is a dead end, avoiding the cost of exploring its successors. On the other hand, a more accurate heuristic is usually more expensive to compute, and this additional time will burden every visited state. Furthermore, the computation of sophisticated heuristics often depends on additional data structures that must be updated during the visit process, adding more time and in some cases more space to the requirements of the algorithm.

In [7] we have presented VF3, a recent algorithm based on this approach, especially devised to be effective on large and dense graphs, which are often problematic for other matching algorithms. The experiments in the paper demonstrate the effectiveness of VF3 in comparison with several recent state-of-the-art algorithms. In [5] we introduced a simplified version of VF3, named VF3-Light, that removes some of the heuristics used in VF3.

* Corresponding author.
E-mail address: vcarletti@unisa.it (V. Carletti).

https://doi.org/10.1016/j.patrec.2019.07.001
0167-8655/

In this paper we present in greater detail the V3-Light algorithm, discussing how the removal of those heuristics, while reducing the ability to prune unfruitful states, leads to an improvement of the per-state visit time, and also allows the algorithm to remove some of the data structures needed for the implementation of VF3. The expectation that this may lead, on some kinds of graphs, to a better overall matching time has been experimentally confirmed by an extended comparison on several publicly available datasets. We have further expanded the experimentation by including a comparison with other state-of-the-art subgraph isomorphism algorithms.

2. Preliminary definitions

Graphs are mathematical abstractions adopted to represent objects in terms of their parts and the relationships among these, namely the nodes and the edges respectively. A graph is so defined as an ordered couple $G = (V, E)$ where V is the set of nodes and $E \subset V \times V$ the set of edges. If edges are considered as ordered pairs, the graph is *directed*; otherwise, if edge (v_i, v_j) is considered equivalent to (v_j, v_i), the graph is *undirected*.

In many cases, graphs carry a semantic information together with the structure represented in terms of node and edge *labels* (or more generally *attributes*). To this aim, two addition sets of node labels $\mathcal{L}_v$ and edge labels $\mathcal{L}_e$ and the related *labeling functions* $\lambda_v : V_1 \cup V_2 \to \mathcal{L}_v$ and $\lambda_e : E_1 \cup E_2 \to \mathcal{L}_e$ are added to the graph: $G = (V, E, \mathcal{L}_v, \mathcal{L}_e, \lambda_v, \lambda_e)$.

Once the basic definitions about graphs are given, let us define what is *graph matching*. In the wider definition, given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, graph matching is the problem of finding a *mapping function* M: $V_1 \to V_2$ that satisfies a set of structural and semantic constraints. Considering this definition, there are several problems that lie into this category (for more details, we suggest to read [9,11,15]). Among them, *subgraph isomorphism* is the problem of finding an exact structural and semantic correspondence between the nodes of a *pattern graph* and a subset of the nodes of a *target graph*. For the sake of simplicity, hereinafter G_1 will be the pattern and G_2 the target.

In this case, the mapping function M must be injective and *structure preserving*:

$$\forall u \in V_1,\ \exists \mu(u) \in V_2 : \lambda_v(u) \approx \lambda_v(\mu(u)) \tag{1}$$

$$\forall u, u' \in V_1,\ u \neq u' \Rightarrow \mu(u) \neq \mu(u') \tag{2}$$

$$\forall (u, u') \in E_1,\ \exists (\mu(u), \mu(u')) \in E_2 : \lambda_e((u, u')) \approx \lambda_e((\mu(u), \mu(u'))) \tag{3}$$

$$\forall (v, v') \in E_2,\ \exists (\mu^{-1}(v), \mu^{-1}(v')) \in E_1 : \lambda_e((v, v')) \approx \lambda_e((\mu(v), \mu(v'))) \tag{4}$$

where $\mu(u)$ represents the node in V_2 that is mapped to $u \in V_1$ by M, and $\mu^{-1}(v) \in V_1$ is the node mapped to $v \in V_2$ by the inverse of M, and $\approx$ denotes the *compatibility* of label values (which, depending on the application, may be a relation less strict than equality).

3. The VF3-Light algorithm

The algorithm we present in this paper is a simplified version of the state of the art algorithm VF3 proposed by [7]. In this Section, we will describe how the algorithm works and the main differences from VF3.

3.1. Problem representation

Being a descendant of VF3, the problem of finding all the possible matching functions satisfying the subgraph isomorphism constraints is faced by VF3-Light using a *State Space Representation*. According to the latter, the algorithm visits a tree-structured search space of states using a depth-first search so as to guarantee a memory complexity that is linear w.r.t. the size of pattern graph. Each state s in this space corresponds to a partial mapping involving a subset of the nodes belonging to pattern. In particular, the states whose partial mapping satisfies the constraints imposed by the subgraph isomorphism are considered to be *consistent*. Leaves of the search space represent complete mappings; if they are also consistent, they represent solutions to the subgraph isomorphism problem (*goal states*).

The search space is structured so that each state s is connected only to its direct descendants obtained by adding a pair of nodes $(u, v) \in V_1 \times V_2$ to the partial mapping of s. It can be demonstrated that inconsistent states will never generate consistent descendants. On the other hands, starting from a consistent state it is possible to generate an inconsistent one. The main difference between VF3 and VF3-Light lies in the set of *feasibility rules* they use to guarantee that only consistent states are explored in the search tree.

3.2. Feasibility rules

The main purpose of the *feasibility rules* is to verify that the addition of a new couple of nodes $(u \in V_1,\ v \in V_2)$ to a consistent state s will generate a new state s' that can potentially lead the algorithm to a goal state. Thus, the consistency of s' is a necessary condition, but VF3 checks additional conditions (*look-ahead conditions*) to detect if s' will not have consistent descendants after one or two search steps.

In details, naming $M(s) \subset V_1 \times V_2$ the partial mapping at the state s and denoting as $M_1(s)$ and $M_2(s)$ the projections of $M(s)$ onto V_1 and V_2 respectively, it can be shown that s is consistent iff the graphs $G_1(s)$ and $G_2(s)$, obtained from the nodes in $M_1(s)$ and $M_2(s)$ respectively, are isomorphic. So, the addition of u to $M_1(s)$ and v to $M_2(s)$ must generate two graphs $G_1(s')$ and $G_2(s')$ that are still isomorphic. In VF3-Light the feasibility rule is composed of two parts:

$$F(s, u, v) = F_s(s, u, v) \wedge F_c(s, u, v) \tag{5}$$

The *semantic feasibility rule* F_s checks if u and v have compatible labels and if the edges connecting them to $M_1(s)$ and $M_2(s)$ also have compatible labels. The *structural feasibility rule* F_c checks that if an edge exists between u and a node in $M_1(s)$, an edge must also exist between v and the corresponding node in $M_2(s)$, and vice versa.

VF3 uses two additional *look-ahead rules* F_{la1} and F_{la2} (see Eq. (6)), to prune consistent states that will demonstrably not have consistent descendants after one (F_{la1}) or two (F_{la2}) steps in the search space:

$$F_{ext}(s, u, v) = F(s, u, v) \wedge F_{la1}(s, u, v) \wedge F_{la2}(s, u, v) \tag{6}$$

These two rules have been demonstrated to be useful to deal with very large graphs, but they require VF3 to compute the following additional structures, for each state s, for both the graphs:

- $\widetilde{\mathcal{P}}_1(s) \subset V_1$ and $\widetilde{\mathcal{P}}_2(s) \subset V_2$, the sets of nodes outside $M(s)$ having an edge whose destination is a node in $M_1(s)$ (for $\widetilde{\mathcal{P}}_1$) or in $M_2(s)$ (for $\widetilde{\mathcal{P}}_2$);
- $\widetilde{\mathcal{S}}_1(s) \subset V_1$ and $\widetilde{\mathcal{S}}_2(s) \subset V_2$, the sets of nodes outside $M(s)$ having an edge whose origin is a node in $M_1(s)$ (for $\widetilde{\mathcal{S}}_1$) or in $M_2(s)$ (for $\widetilde{\mathcal{S}}_2$).

The sets $\widetilde{\mathcal{S}}(s_n)$ and $\widetilde{\mathcal{P}}(s_n)$ require an overall memory occupation of $O(N_2)$, where N_2 is the size of G_2, thanks to the depth-first search and to the fact that VF3 reuses the data structures of the parent state when a descendant is generated from it. Nevertheless, the time required to compute these structures is not negligible; indeed, at each state s, it is proportional to the degrees of u and v. Moreover, the same time is spent when the algorithm has finished to visit a state and has to restore the structures to their previous content.

Therefore, there is a real advantage in using the look-ahead rules when the overall amount of time required to manage the structures and compute the rules at each state is balanced by the reduction in the number of states that are explored by the algorithm. In some cases, such as on small graphs, the use of these rules may cause a decrease in performance.

3.3. Algorithm overview

In Fig. 1 we show the structure of VF3-Light. Before starting the visit, actually realized by the procedure `Match`, the algorithm performs some preliminary steps aimed at precomputing some information used during the visit.

The procedure `ComputeOrdering` is used to sort the nodes of the pattern graph in order to start the exploration from the most rare and constrained nodes. The result of this procedure is a *node exploration sequence* N_{G_1} where the nodes $u \in V_1$ are arranged according to the following criteria, ordered by their priority:

- *Most constrained first*: highest number of connections with the nodes that have already been inserted in N_{G_1}, since each connection is a constraint in the mapping.
- *Rarest first*: lowest probability of finding a node $v \in V_2$ that has the same label and a equal or higher degree.
- *Highest degree first*: the higher is the degree of a node, the more constraints will introduce in the mapping.

The sequence N_{G_1} is then used by the algorithm to visit the pattern graph. It is worth to note that the sequence guarantees that the search space is structured as tree and not as graph; each position of the sequence correspond to a specific level of the tree where the node to match, for the pattern graph, is given by the sequence and cannot be encountered twice going down the space, thus avoiding the possibility to have loops. In addition, the exploration sequence makes the algorithm insensitive w.r.t. the permutation of the pattern graph and the sequence allows to compute, for the pattern, some structures required by the visit before it starts. In particular, this last optimization is performed by the procedure `PreprocessPatternGraph`. As it is possible to note in Fig. 1, such a procedure prepares the first state of the search space, namely s_0, together with an additional structure, the `Parent` tree, that links each node of V_1 to the first node in N_{G_1} that is connected to it. The latter is used during the visit to boost the search for the nodes, belonging to the target graph, that are suitable to generate new states.

After the preprocessing step, the recursive depth-first visit starts as shown in Fig. 2. For each state s the algorithm evaluates all the possible candidate couples to generate a new state

```
function VF3-Light(G1,G2)
   N_G1 := ComputeOrdering(G1,G2)
   s0, Parent := PreprocessPatternGraph(G1,N_G1)
   Results := {}
   Match(s0, G1, G2, N_G1, Parent, Results)
   return Results
end
```

Fig. 1. Outline of the algorithm.

```
procedure Match(s, G1, G2, N_G1, Parent, out Results)
   if IsGoal(s) then append M(s) to Results
   else
      for (un,vn) ∈ NextCandidates(s, N_G1, Parent, G1,
G2)
         if IsFeasible(s, un, vn) then
            sn := ExtendState(s, un, vn)
            Match(sn, G1, G2, N_G1, Parent, Results)
            RestoreState(s, un, vn)
         end if
      end for
   end if
end
```

Fig. 2. The recursive Match procedure. If only the first solution is desired, the procedure is slightly different and stops when the first goal state is found. Here s is the current state, while s_n is a new state obtained adding the couple (u_n, v_n) to the current partial mapping $M(s)$.

s_n. Among all the couples, only those that are feasible are considered to generate a new state (procedure `ExtendState`) and so proceed with the recursive search. The consistency check is performed by the procedure `IsFeasible` that verifies if the rules in Eq. (5) are satisfied. When all the children of a state have been explored, the algorithm backtracks (procedure `RestoreState`). The recursive call stops as a goal state is found or when no more candidates are available for the current state s. The recursive procedure shown in Fig. 2 is used to search for all the possible solutions. In case only the first solution is desired the algorithm procedure stops when the first goal state is found or when no more states have to be explored (i.e. there are no existing solutions).

As for all the subgraph isomorphism algorithms in the worst case, i.e. when dealing with symmetric graphs, the time complexity is exponential. Concerning the average case, since the overall structure of VF3-Light is similar to that of VF3, it is possible to refer to the analysis proposed by [7]; the only difference lies into the coefficients of the polynomial. For the same reason, the space complexity of VF3-Light is still linear w.r.t. the size of G_2.

4. Experiments

In order to verify the effectiveness of the proposed approach, we tested our method over a large number of datasets, including different typologies of graphs, and representative of different applications (such as biological and social networks). Extensive testing is a very important and not negligible task in the graph matching field, since there is no single algorithm working better than the others on all the kind of graphs and for all the applications. Thus, it is important to understand the situations in which the proposed algorithm performs better and is suggested to be used. To this aim, in our benchmark, we have considered five datasets composed of a very wide variety of graphs. Firsly we used three well-known datasets: MIVIA proposed by [10], MIVIA-Complementary proposed by [4] (named MIVIA-C), MIVIA Biological Graphs proposed by [8] (named MIVIA-B). The former is a synthetic dataset composed of 56,400 couples of unlabeled graphs belonging to the following families: bounded valence, random graphs and open meshes (regular and irregular). MIVIA-C is composed of 1,258,543 couples of unlabeled graphs randomly extracted from the MIVIA dataset among those have not matching solutions; it has been introduced in order to analyze the performance of the algorithms when no solutions exist. MIVIA-B as been introduced during the First International Contest on Pattern Search in Biological Graphs by [8]; it is composed of real graphs extracted from Proteins, Contact Maps and Molecules. Finally, two additional datates, proposed in [13], have been taken into accout: CVIU11, is composed of unlabeled graphs extracted from segmented images, and PR15 com-

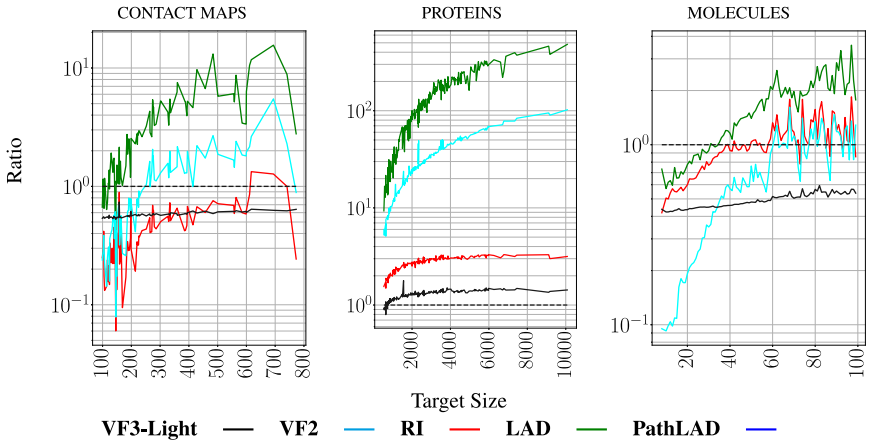

Fig. 3. The ratio between the time of each algorithm and VF3, that we used as a baseline, for the MIVIA-B dataset.

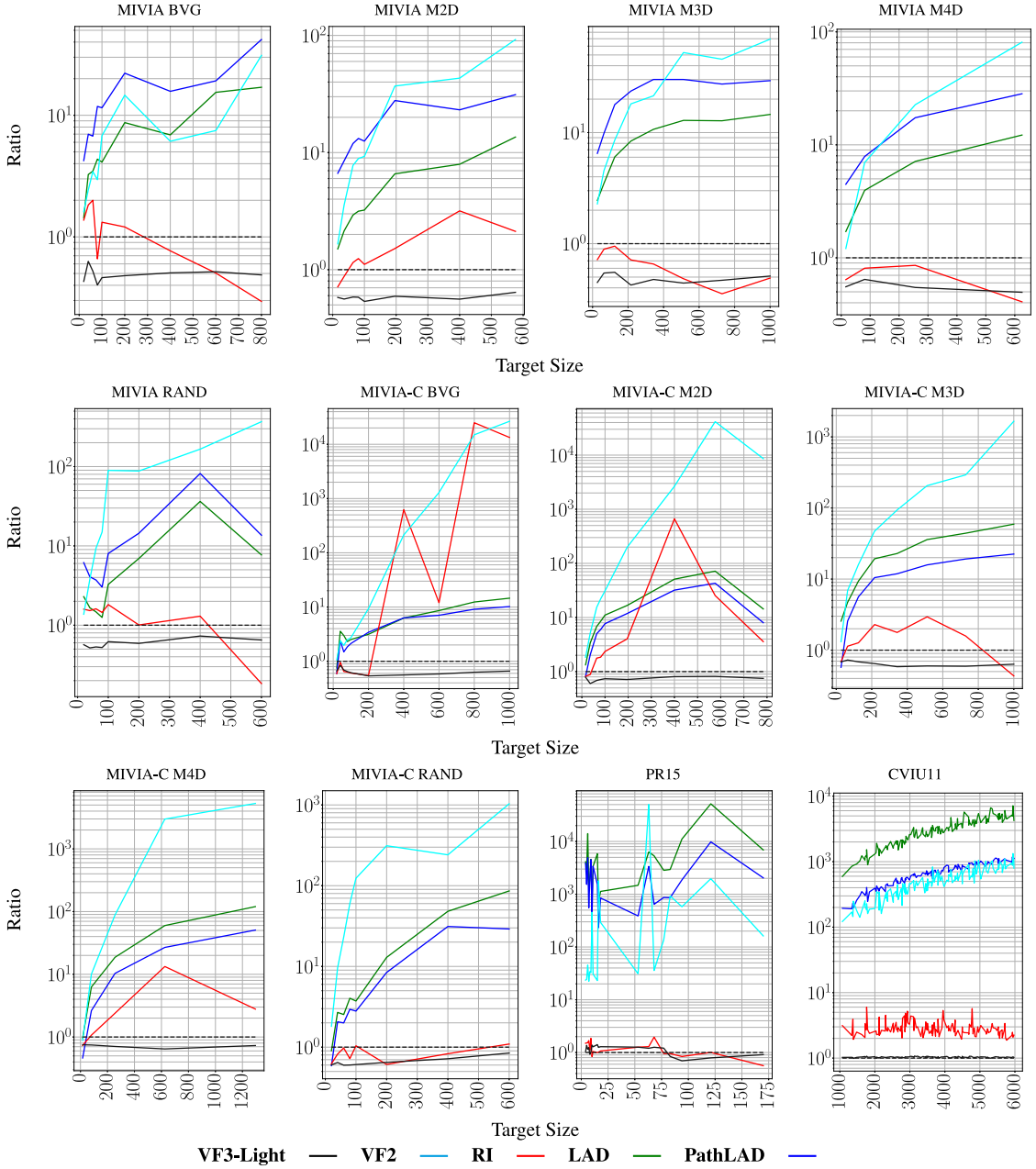

Fig. 4. The ratio between the time of each algorithm and VF3, that we used as a baseline.

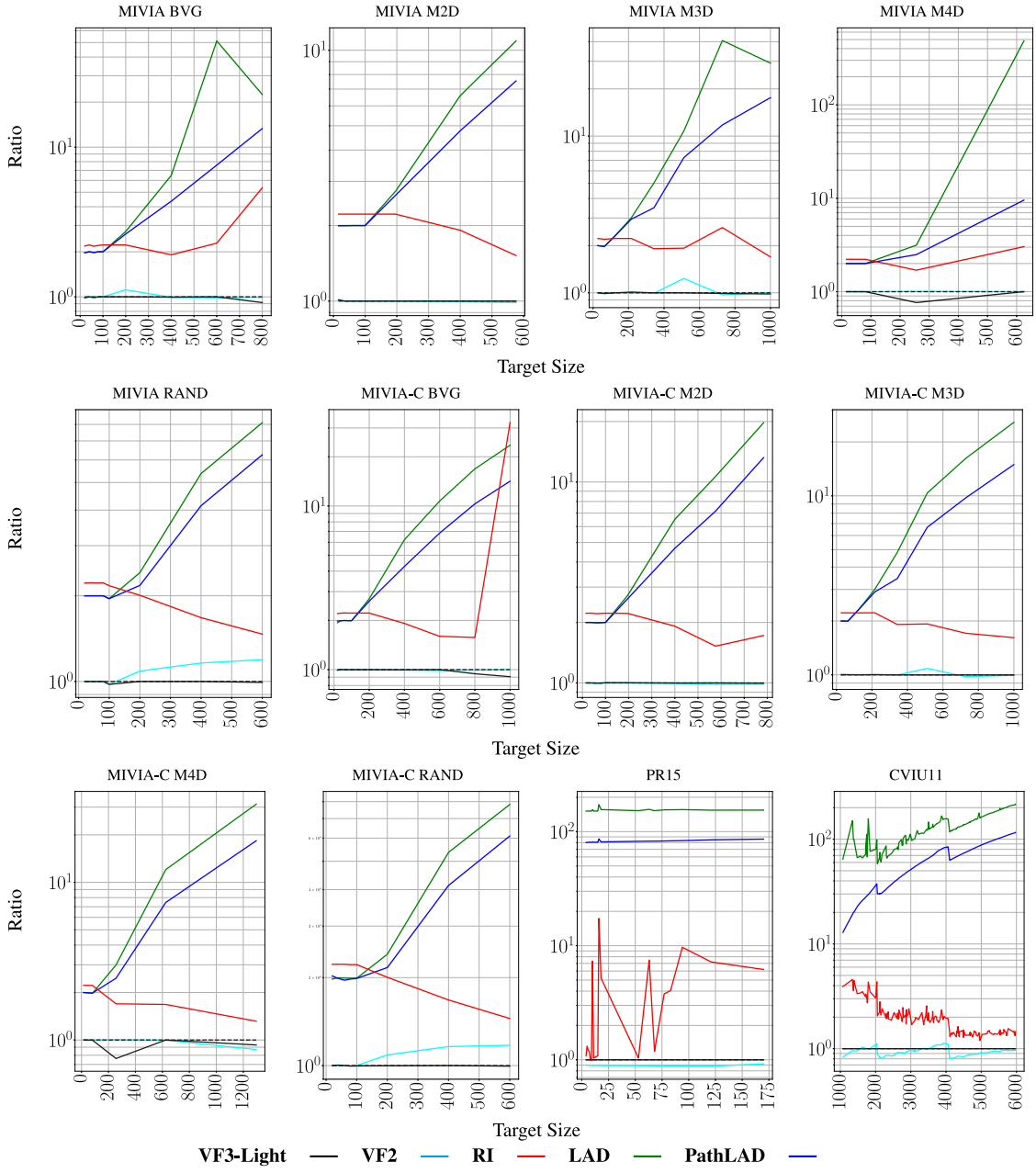

Fig. 5. The ratio between the memory usage of each algorithm and VF3, that we used as a baseline.

posed of 24 pattern graphs and one target graph of 4838 nodes, extracted from segmented images.

The experimentation has been conducted by using a cluster infrastructure running simultaneously 40 different virtual machines, hosted by VMWare ESXi 5 and provided with the Linux operating system. Each virtual machine has been provided with two dedicated AMD Opteron 6376 processors running at 2300 MHz, with 2Mb of cache and 4Gb of RAM. Each virtual machine has been also configured so as to avoid time drifting or lost clocks, that could have potentially introduced some bias in the time measurement.

The effectiveness of the proposed method has been evaluated by considering the time required to find all the solutions and the memory required for the matching. It is worth noting that we have not considered the accuracy because we are using exact graph matching algorithms, thus they provide the exact solution if it exists. However, we have checked that the reported solutions were correct. As already done in [6], we evaluate the memory requirements by using the process statistics provided by the operating system; since Linux-based operating system pre-allocates some memory pages before they are actually required, this can be considered only partially accurate for small graphs, but it is enough accurate for our purpose.

The following algorithms have been used for the benchmark: VF3, proposed in [7], which inspires the proposed VF3-Light; VF2, proposed in [10], the tree search based approach inspiring VF3 and being among the fastest algorithm available in the literature for more than ten years; RI, proposed by [3], a three-search based algorithm which shares with VF3-Light the property of avoiding look-ahead, but is based on different heuristics and a different procedure for sorting; DirectedLAD and Path LAD, proposed in [12], based on a constraint programming approach.

The obtained results have been reported in Figs. 3 and 4: in more details, we report in each figure the matching time of each algorithm, normalized with respect to the time required by VF3,

that we considered as our baseline approach. It implies that VF3 is represented by a constant line at 1, while for instance 1.3 means a matching time 30% slower than VF3.

Analyzing the results, we can note that in the case of matching large graphs (up to 10,000 nodes), with many solutions, VF3 is still the fastest algorithm available in the literature. This is evident on the Protein graphs of the MIVIA-B dataset (see Fig. 3), where the average number of solutions is 46. On problems with fewer solutions, and when dealing with smaller graphs, the effectiveness of the proposed VF3-Light emerges. For instance, on the Molecules and Contact map graphs of the MIVIA-B dataset, VF3-Light is definitely faster than VF3 and competes for the first position with RI. It is important to highlight that the average number of solutions for Molecules and Contact maps is 8 and 7, respectively.

A similar behavior occurs when examining the CVIU11 dataset, where the performance of VF3 and VF3 Light are almost comparable; this is due to the fact that the graphs in this dataset have a bounded valence (number of edges per node), and a large size. On smaller graphs of the same type (such as in BVG graphs of the MIVIA dataset) the advantages deriving from VF3-Light are evident. Note that on Random graphs there is an anomaly at 600 nodes: it happens because a single pattern/target couple is quite problematic for both VF3 and VF3-Light and makes the average matching time for both the algorithms quite long.

The same trend is also confirmed by looking at the results achieved on the whole MIVIA dataset: indeed, with small to medium-sized graphs (up to about 500 nodes) the proposed VF3-Light algorithm shows the best performance, while for higher dimensions it competes in some cases with RI, which avoids look-ahead as well, and thus is designed for managing the same simple situations for which VF3-Light is thought. Anyway this is still better than the original VF3 in all the situations. On the whole MIVIA-C dataset, VF3-Light still confirms to be the fastest of the tested algorithms, being able to quickly verify the absence of the matching.

Finally, it is also important to highlight that the advantages introduced by VF3-Light from a computational point of view are not paid in terms of memory requirements. In fact, as we can note from Fig. 5, although the memory occupation of VF3 was already very good, VF3-Light is almost always the algorithm requiring less memory (only in two of the tests it was outperformed by RI on this regard).

5. Conclusions

In this paper we have introduced VF3-Light, a subgraph isomorphism algorithm obtained by removing some of the heuristics used in VF3, the so called look-ahead functions. The removal of these heuristics makes the algorithm faster in the visit of each search state, but also implies that a larger number of states may need to be visited for finding the solutions. An experimental evaluation on several kinds of graphs shows that indeed on very large or very dense graphs, for which the VF3 algorithm was designed, the look-ahead heuristics give an advantage, but on other, simpler kinds of graphs VF3-Light is able to outperform VF3. In our experiments we have also compared VF3-Light with other state-of-the-art algorithms; the results demonstrate that the proposed algorithm is often the fastest one, and in most cases it is the one with the smaller memory requirements.

Conflict of interest

None.

References

[1] I. Almasri, X. Gao, N. Fedoroff, Quick mining of isomorphic exact large patterns from large graphs, in: IEEE International Conf. on Data Mining Workshop, 2014, pp. 517–524.
[2] V. Bonnici, R. Giugno, On the variable ordering in subgraph isomorphism algorithms, IEEE/ACM Trans. Comput. Biol. Bioinform. 14 (1) (2017) 193–203.
[3] V. Bonnici, R. Giugno, A. Pulvirenti, D. Shasha, A. Ferro, A subgraph isomorphism algorithm and its application to biochemical data, BMC Bioinform. 14 (2013).
[4] V. Carletti, P. Foggia, A. Greco, A. Saggese, M. Vento, Comparing performance of graph matching algorithms on huge graphs, Pattern Recognit. Lett. (2018), doi:10.1016/j.patrec.2018.06.025.
[5] V. Carletti, P. Foggia, A. Greco, A. Saggese, M. Vento, The VF3-Light subgraph isomorphism algorithm: when doing less is more effective, in: Joint IAPR International Workshop, S+SSPR 2018, Beijing, China, August 17–19, 2018, pp. 315–325. Proceedings
[6] H. Bunke, M. Vento, Benchmarking of graph matching algorithms, in: Proc. of the 2nd Workshop on Graph-based Representations, 1999.
[7] V. Carletti, P. Foggia, A. Saggese, M. Vento, Challenging the time complexity of exact subgraph isomorphism for huge and dense graphs with vf3, IEEE Trans. Pattern Anal. Mach. Intell. 40 (4) (2018) 804–818, doi:10.1109/TPAMI.2017.2696940.
[8] V. Carletti, P. Foggia, M. Vento, X. Jiang, Report on the first contest on graph matching algorithms for pattern search in biological databases, in: GbR2015, 2015, pp. 178–187.
[9] D. Conte, P. Foggia, C. Sansone, M. Vento, Thirty years of graph matching in pattern recognition, Int. J. Pattern Recognit. Artif. Intell. 18 (3) (2004) 265–298.
[10] L. Cordella, P. Foggia, C. Sansone, M. Vento, A (sub)graph isomorphism algorithm for matching large graphs, IEEE T Pattern Anal. 26 (2004) 1367–1372.
[11] P. Foggia, G. Percannella, M. Vento, Graph matching and learning in pattern recognition on the last ten years, Int. J. Pattern Recognit. Artif. Intell. 28 (1) (2014).
[12] L. Kotthoff, C. Mccreesh, C. Solnon, Portfolios of subgraph isomorphism algorithms, in: Learning and Intelligent OptimizatioN Conference (LION 10), Springer, 2016.
[13] C. Solnon, Solnon datasets, 2017, http://liris.cnrs.fr/csolnon/SIP.html.
[14] J.R. Ullmann, An algorithm for subgraph isomorphism, J. Assoc. Comput. Mach. 23 (1976) 31–42.
[15] M. Vento, A long trip in the charming world of graphs for pattern recognition, Pattern Recognit. (2014) 1–11.