VF2 Plus: An Improved version of VF2 for Biological Graphs

Vincenzo Carletti(✉), Pasquale Foggia, and Mario Vento

DIEM, Department of Information Engineering,
Electrical Engineering and Applied Mathematics,
University of Salerno, Salerno, Italy
{vcarletti,pfoggia,mvento}@unisa.it
http://mivia.unisa.it

Abstract. Subgraph isomorphism is a common problem in several application fields where graphs are the best suited data representation, but it is known to be an NP-Complete problem. However, several algorithms exist that are fast enough on commonly encountered graphs so as to be practically usable; among them, for more than a decade VF2 has been the state of the art algorithm used to solve this problem and it is still the reference algorithm for many applications. Nevertheless, VF2 has been designed and implemented ten years ago when the structural features of the commonly used graphs were considerably different. Hence a renovation is required to make the algorithm able to compete in the challenges arisen in the last years, such as the use of graph matching on the very large graphs coming from bioinformatics applications. In this paper we propose a significant set of enhancements to the original VF2 algorithm that enable it to compete with more recently proposed graph matching techniques. Finally, we evaluate the effectiveness of these enhancement by comparing the matching performance both with the original VF2 and with several recent algorithms, using both the widely known MIVIA graph database and another public graph dataset containing real-world graphs from bioinformatics applications.

1 Introduction

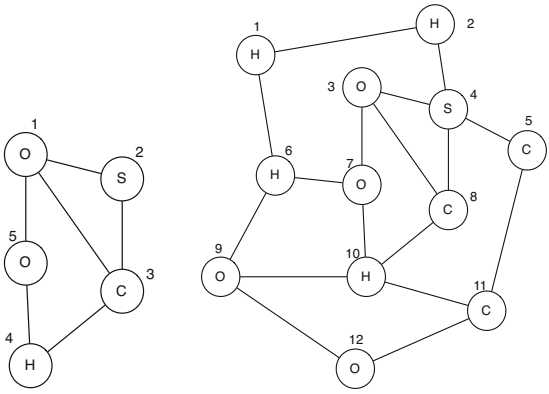
In several application fields graphs are the best suited data structure to represent entities and relationships composing the problem domain. Think, as examples, to bioinformatics or chemoinformatics [2,9,10], where proteins and drugs structures, metabolic networks, interaction networks and so on, are naturally represented using graphs. Other noteworthy examples are social network analysis, where the social interactions are represented by graphs of people, and the semantic web, where the Resource Description Framework (RDF) defines a standard graph format to represent the relationships existing among a set of web resources. In many of these applications, a relevant problem is the search for a pattern graph within a target graph, namely the subgraph isomorphism problem [3,6,18]. This problem is known to be NP-Complete in the worst case, i.e. when the two graphs

C.-L. Liu et al. (Eds.): GbRPR 2015, LNCS 9069, pp. 168–177, 2015.
DOI: 10.1007/978-3-319-18224-7_17

are complete or in general symmetric. Thus, the only way to solve the subgraph isomorphism problem in a reasonable time is to exploit the knowledge on the structure of the graphs. Due to this fact, many algorithms have been proposed during the last decade. In order to provide a complete overview of the current state of the art we identified three main trends: tree search based algorithms, constraint programming algorithms and index based algorithms. The first trend comprises algorithms coming from the field of artificial intelligence; these algorithms deal with the problem by searching the solution inside a state space, generally using a depth-first search Ullmann [16] and VF/VF2 [4] are two very popular algorithms adopting this paradigm. Recently, new algorithms tried to refine this approach for specific kinds of graphs, as instance RI, RIDS [1] and L2G. The constraint programming approach [20] deals with subgraph isomorphism by using a diametrically opposite method, with respect to the previous one. Indeed, the rationale is to filter, among all the possible couples of nodes, those that are surely not contained in a complete matching solution. The algorithm by McGregor [12] is an early example of this trend, that has been followed during the last decade by Larrosa and Valiente[11], ILF [20], LAD [15] and the last algorithm by Ullmann [17]. The last trend comprises algorithms that extend the pure database approach, i.e. graph indexing, where the main problem is just to determine if a pattern graph is inside a target graph, without identifying which subgraphs in the target graph are isomorphic with the pattern. These algorithms exploit a set of features to define an index for the pattern graph structure, used for driving the search for possible solutions on the target. Recent algorithms following this trend are: GraphQL [8], QuickSI [14], GADDI [21], SPath [22] and TurboIso [7]. The great variety of algorithms is due to the wide diversity of problems, and so of graph representations, in which the subgraph isomorphism is employed. Among these it is important to note that VF2 is one of the most employed algorithm for graph and subgraph isomorphism due to its flexibility [2] in addressing different exact graph matching problems. However, VF2 has been proposed more than ten years ago, when the main applications of exact graph matching were fingerprint recognition, character recognition and molecular structure comparison; thus, the structural features of the graphs employed then were considerably different from the current one, think as example to the size. In this paper, we propose a set of improvements to the VF2 algorithm for large biological graphs and we call the improved version VF2 Plus. Finally, in order to provide an assessment of the improvements, we compared VF2 Plus with VF2, RI, L2G and LAD on the subgraph isomorphism problem.

2 Preliminaries

Before starting the description of the improvements in VF2 Plus, it is important to provide an overview of VF2. As introduced in Section 1, VF2 [4] is a well known state of art algorithm able to deal with different exact graph matching problems, based on a state space representation: each state is a partial mapping between the two given graphs, while goal states are complete mappings consistent with the constraints of the problem we are facing. Hence, the search space



(a) Pattern Graph G_1 (b) Target Graph G_2

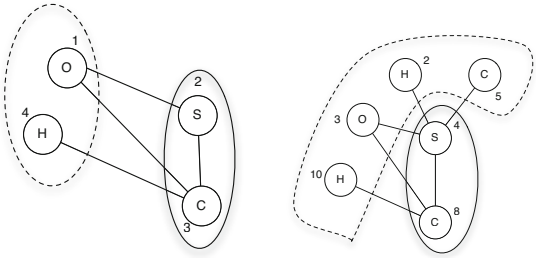
Fig. 1. Graphs used as example to detail how VF2 and VF2 Plus work

is explored using a depth-first strategy with backtracking, driven by a set of feasibility rules to prune, in advance, unfruitful search paths.

In order to provide more details, let us introduce some notations that will be used in the following. Let be $G_1 = (V_1, E_1)$ the pattern graph (Figure 1(a)), and $G_2 = (V_2, E_2)$ the target graph (Figure 1(b)), being V_1, V_2 and E_1, E_2 respectively the vertices and the edges sets of each graph. Let us define the size of the two graphs as the number of the nodes inside the sets V_1 and V_2, so that $|G_1| = |V_1|$ and $|G_2| = |V_2|$.

The algorithm has to keep track, at each state, of the vertex pairs inside the current mapping and of the set of nodes where to search for the next candidate pairs. As shown in Figure 3, each state s stores mapped couples in a set $M(s)$, namely the core set, and uses vertices in two separated sets $M_1(s)$ and $M_2(s)$, respectively for first and second graph. Then, for each vertex in $M_1(s)$ and $M_2(s)$, neighbors not yet in the core set are kept into two sets $T_1(s)$ and $T_2(s)$, called terminal sets. Obviously, in the case of directed graph, each states uses two different pairs of terminal sets: $T_1^{in}(s)$, $T_2^{in}(s)$ for neighbors connected by incoming edges and $T_1^{out}(s)$, $T_2^{out}(s)$ for these connected by outgoing edges. Despite this representation seems to need a space complexity quadratic with respect to the number of the states, VF2 exploits an optimized representation able to reduces this complexity to linear.

Before generating a new state, VF2 selects the next candidate pair (u, v) by picking u from $T_1(s)$ and v from $T_2(s)$ if $T_1(s) \neq 0$ and $T_2(s) \neq 0$, or from the sets $\widetilde{N_1}(s)$, $\widetilde{N_2}(s)$ that contain remaining vertices of G_1 and G_2, not in the core or in the terminal sets:



(a) Pattern Graph G_1 (b) Target Graph G_2

Fig. 2. Core sets (solid line) and terminal sets (dotted line) used by VF2 to represent the state of the current matching and the next candidates set. In the example we can identify the following sets: $M(s) \equiv \{(3,8),(2,4)\}$, $M_1(s) \equiv \{3,2\}$, $M_2(s) \equiv \{8,4\}$, $T_1(s) \equiv \{1,4\}$, $T_2(s) \equiv \{2,3,5,10\}$.

$$P_1(s) = \begin{cases} T_1(s), & \text{if } |T_1(s)| \neq 0 \wedge |T_2(s)| \neq 0 \ , \\ \widetilde{N_{1p}}(s), & \text{otherwise} \end{cases} \tag{1a}$$

$$P_2(s) = \begin{cases} T_2(s), & \text{if } |T_1(s)| \neq 0 \wedge |T_2(s)| \neq 0 \ , \\ \widetilde{N_{2p}}(s), & \text{otherwise} \end{cases} \tag{1b}$$

After that the candidate pair (u, v) has been selected, VF2 evaluates if it is certainly not moving towards a solution by using a set of feasibility rules:

Core Rule. The main rule needed to solve the specific problem we are facing; it guarantees to generate only consistent states, i.e. states wherein the partial mapping can possibly be part of a solution.

Look-Ahead Rules. Secondary rules used to search for possible inconsistencies, in order to detect early states without any goal descendants.

In order to understand how feasibility rules work, let us consider the problem of subgraph isomorphism on undirected graphs G_1 and G_2, as shown in Figure 1; VF2 uses the following rules:

$$\begin{aligned} & R_{core}(s,u,v) \iff \\ & \forall u' \in adj(u) \cap M_1(s) \exists! v' \in adj(v) \cap M_2(s) : (u',v') \in M(s) \\ & \wedge \forall v' \in adj(v) \cap M_2(s) \exists! u' \in adj(u) \cap M_1(s) : (u',v') \in M(s) \end{aligned} \tag{2a}$$

$$R_{term}(s,u,v) \iff |adj(u) \cap T_1(s)| = |adj(v) \cap T_2(s)| \tag{2b}$$

$$R_{new}(s,u,v) \iff |adj(u) \cap \widetilde{N_1}(s) = |adj(v) \cap \widetilde{N_2}(s)| \tag{2c}$$

The effectiveness of the previous rules in terms of search space reduction and complexity has been well discussed in a recent paper by N. Dahm and H. Bunke [5].

However, the exponential nature of the subgraph isomorphism problem and the wide variety of application fields raise the need for an algorithm that is generic with respect to the specific contexts, but easy to specialize, by using simple heuristics if possible, in order to reduce the explored search space and consequently the time to find the solutions.

3 Improvements

Once we described how VF2 works, it is easier to discuss the improvements provided. In particular, VF2 Plus improves two important weaknesses of VF2: the total order relationships on the nodes of the pattern graph and the structure of the terminal sets.

3.1 Total Order Relationship

The first weakness of VF2 lies in the total order relationship employed. Such order relationship, defined on the nodes of the pattern graph, is very simple and provides no real advantage to the algorithm other than producing a search space structured as a tree. In fact, if the search space were a graph then a state with k different pairs could be reached from $k!$ different paths, making the search unfeasible. However, if well defined, a total order relationship can give an additional advantage by providing a sequence of nodes of the first graph that makes the algorithm able to explore the search space more efficiently. Thus, VF2 Plus uses a sorting procedure that prefers nodes, on the pattern graph, with the lowest probability to find a candidate on the target graph and the highest number of connections with the nodes already used by the algorithm. The procedure provides a sorted node sequence that imposes to the algorithm how to explore the pattern graph. For this reason, once the sequence is given the algorithm already knows which node of the pattern graph it is going to select as candidate at each level of the search space. Indeed, it is trivial to understand that all of the states at the same level share the same candidate node on the pattern graph. In this way, VF2 Plus is able to compute the state of the terminal sets related to the pattern graph at each level before staring the matching.

Evaluate the Probability. On subgraph isomorphism a node $v \in G_2$ is feasible with a node $u \in G_1$ if they share the same label, $lab(u) = lab(v)$, and their degree is compatible, $deg(u) \leq deg(v)$. Thus, both semantic and structural node features, i.e. the label and the degree, can be employed to evaluate the probability to find a candidate $v \in G_2$ for $u \in G_1$.

Firstly, the procedure computes the frequency of each label and degree on G_2. So, let us define:

- $P_{lab}(L)$: a priori probability to find a node with label L on G_2.
- $P_{deg}(d)$: a priori probability to find a node with degree d on the G_2.

For each pattern node $u \in G_1$ the probability is computed as follows:

$$P(u) = P_{lab}(L) * \cup_{d' \geq d} P_{deg}(d') \tag{3}$$

Order Relationships. Once the probability is computed, the algorithm begins the ordering procedure. Let's define the set of already sorted nodes M, T the set of nodes candidate to be selected and *degreeM*, of a node in $u \in T$, as the number of edges that are connected to nodes in M. Then the algorithm proceeds as follows:

1. Select the node with the lowest probability.
   (a) If more nodes share the same probability, select the one with maximum degree.
   (b) If more nodes share the same probability and they have the maximum degree, select the first.
2. Put the selected node in the set M and all of its neighboors in the set T.
3. From the set T select the node with the maximum *degreeM*.
   (a) If more nodes share the same *degreeM*, select the one with the lowest probability.
   (b) If more nodes share the same probability and the same *degreeM*, select the first.
4. Go to 2

3.2 Terminal Sets

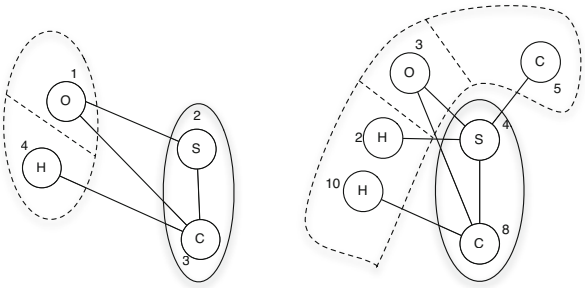
The second critical point lies in the terminal sets, T_1 and T_2. As shown in the Equations (1) (2), these sets are fundamentals both for generating new candidate pairs and for applying the look-ahead rules. But, while for the latter VF2 analyzes just the neighborhood of both the nodes in the candidate pair (u, v), for the former it could explore the terminal sets completely. Thus, since the search for a new candidate pair is performed repeatedly at each state, it is clear that the size of the terminal sets affects the efficiency of VF2. The growth of the sets depends on the average degree, the clustering coefficient and the size of the analyzed graphs. As example, a very bad case is represented by large sparse graphs, such as protein graphs. In these situations VF2 explores more useless candidate nodes inside the terminal sets, while the part of the set that contains feasible candidates is very small. VF2 Plus reduces significantly the explored part of the terminal set by using a new procedure of candidate selection based on two main improvements. First, in order to find a node in G_2 to map to the node $u \in G_1$, the new procedure analyzes just the neighborhood of the node $v' \in G_2$ that is already mapped to a neighbor u' of u. If there are no neighbors of u in the core set, then the pair will be searched outside the terminal sets. Second, before starting the matching, VF2 Plus applies a node classification procedure on the nodes of both the graphs. The aim of this classification is to divide the terminal sets in different subsets, one for each class, and so reduce the number of feasible candidate pair and make the look-ahead rules stronger as applied to each class subset. Indeed, given a set C of n non overlapped class C_i with $i \in [0, n] \subset \mathbb{N}$, the terminal sets can be structured as:

$$\begin{aligned} T_1(s) &\equiv T_1^{C1}(s) \cup T_1^{C2}(s) \cup \cdots \cup T_1^{Cn}(s) \\ T_2(s) &\equiv T_2^{C1}(s) \cup T_2^{C2}(s) \cup \cdots \cup T_2^{Cn}(s) \end{aligned} \tag{4}$$

And the look-ahead rule can be rewritten as follows:

$$R_{ahead}(u,v,s) \iff |adj(u) \cap T_1^{C1}(s)| = |adj(u) \cap T_2^{C1}(s)| \wedge |adj(u) \cap T_1^{C2}(s)| = |adj(u) \cap T_2^{C2}(s)| \wedge |adj(u) \cap T_1^{C3}(s)| = |adj(u) \cap T_2^{C3}(s)| \wedge \cdots \wedge |adj(u) \cap T_1^{Cn}(s)| = |adj(u) \cap T_2^{Cn}(s)| \quad (5)$$

In many real application is possible to identify a suited class function that uses semantic and structural features to classify the nodes. It is important to note that if two nodes of G_1 and G_2 are feasible for the isomorphism they must be in the same class, while the inverse is not true. Thus, once selected a candidate pair from the same class set in T_1 and T_2 is necessary to evaluate the feasibility.



(a) Pattern Graph G_1 (b) Target Graph G_2

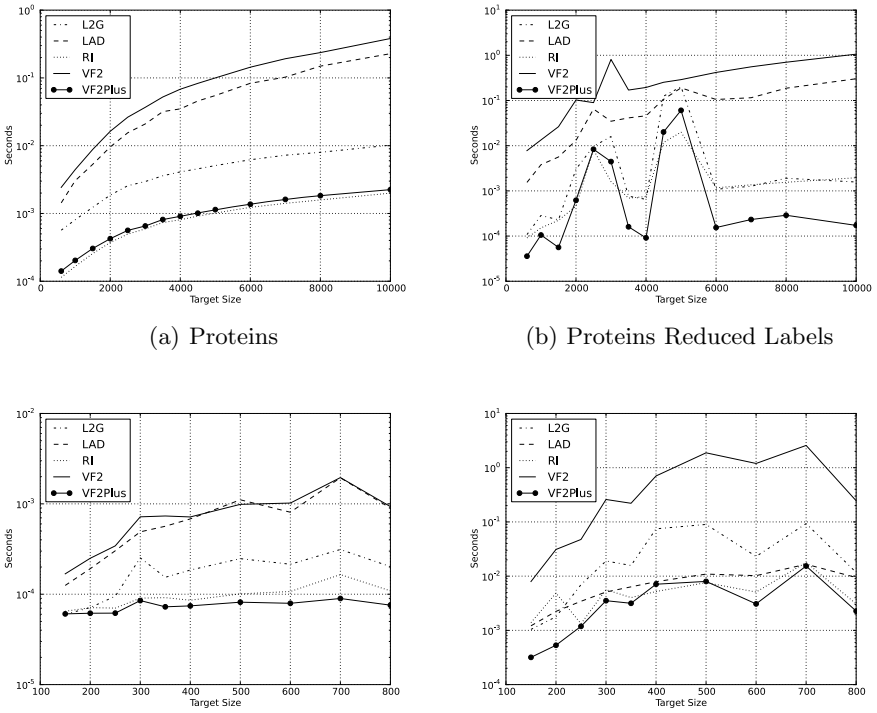
Fig. 3. Core sets (solid line) and terminal sets (dotted line) partitioned in different classes. In this example each node with the same label belongs to the same class.

4 Experiments

In order to prove the effectiveness of the proposed algorithm, we compared VF3 with VF2 and other state of the art algorithms: RI, LAD, L2G. The experiments have been conducted on a linux based system (kernel version 3.13) with an Intel Xeon Twelve Core 3.20 Ghz and 32 GBytes of RAM. We have considered two different parameters: the time to find the first matching solution and the time to find all the matching solutions. For each parameter, the measure has been performed several times on the same pair of graphs to avoid caching effects. The comparison has been performed employing the biological datasets proposed by a recent International Contest on Pattern Search in Biological Databases [19]. The datasets are composed of Contact Map and Protein graphs extracted from the Protein Data Bank [13]. Moreover, we have also considered a modified version of the previous datasets wherein the number of solutions, for each pair of graphs, has been increased by reducing the variability of the labels; we called this dataset *Reduced Labels*. The protein dataset is composed of very large and sparse graphs from 500 to 10000 nodes, with average degree of 4; while the contact maps dataset

contains medium size and dense graphs from 150 to 800 nodes and average degree of 20. In the original version the two datasets presents respectively 6 and 20 different labels, but in the modified version the variability of the labels has been reduced to 4 and 7.

As shown in Figure 4, VF2 Plus always outperforms VF2, LAD and L2G. Especially on very large and sparse graphs the difference, between VF2 and VF2 Plus, is more than two order of magnitude on the time to find all the solutions. Furthermore, the Table 1 shows that VF2 Plus most often equals RI, the algorithm winner of the International Contest on Pattern Search in Biological Databases. Indeed, in several cases, the gap between VF2 Plus and RI is less then the 10% on the matching time (Table 1). However, VF2 Plus performs better than RI for very large and sparse graphs when the number of solutions increases (Figure 4(b)) and for dense medium graphs when the number of label is higher (Figure 4(c)). In both cases, the advantage is mainly given by the class-based terminal sets and the new candidate selection procedure that significantly reduce the number of unfeasible pairs explored.



(a) Proteins

(b) Proteins Reduced Labels

(c) Contact Maps

(d) Contact Maps Reduced Labels

Fig. 4. Time in seconds to find all matchings on the four considered datasets

Table 1. The Tables show the results of the experiments on four biological datasets: Proteins Original, Proteins with reduced labels, Contact Maps Original and Contact Maps with reduced labels. For each dataset two different parameters have been considered: the time to found the first matching (First) and the time to found all the matchings (All). Each cell contains the algorithm with the best time, if the gap between two algorithms is less than the 10% both are provided.

Proteins				
	Original		Reduced Labels	
Size	First	All	First	All
600	VF2P/RI	VF2P/RI	VF2P	VF2P
1000	VF2P/RI	VF2P/RI	VF2P	VF2P/RI
1500	VF2P/RI	VF2P/RI	VF2P	VF2P
2000	VF2P/RI	VF2P/RI	VF2P	VF2P/RI
2500	VF2P/RI	VF2P/RI	VF2P	VF2P/RI
3000	VF2P/RI	VF2P/RI	VF2P	VF2P/RI
3500	VF2P/RI	VF2P/RI	VF2P	VF2P
4000	VF2P/RI	VF2P/RI	VF2P	VF2P
4500	VF2P/RI	VF2P/RI	VF2P	RI
5000	VF2P/RI	VF2P/RI	VF2P	VF2P
6000	VF2P/RI	VF2P/RI	VF2P	VF2P
7000	VF2P/RI	VF2P/RI	VF2P	VF2P
10000	VF2P/RI	VF2P/RI	VF2P	VF2P

Contact Maps				
	Original		Reduced Labels	
Size	First	All	First	All
150	VF2P/RI	VF2P/RI	VF2P	VF2P
200	VF2P/RI	VF2P/RI	VF2P	VF2P
250	VF2P/RI	VF2P/RI	VF2P/RI	VF2P/RI
300	VF2P/RI	VF2P/RI	VF2P/RI	VF2P/RI
350	VF2P/RI	VF2P	VF2P/RI	VF2P/RI
400	VF2P/RI	VF2P	VF2P/RI	VF2P/RI
500	VF2P/RI	VF2P	VF2P/RI	VF2P/RI
600	VF2P	VF2P	VF2P	VF2P/RI
700	VF2P	VF2P	VF2P	VF2P/RI
800	VF2P	VF2P	VF2P	VF2P/RI

5 Conclusions

In this paper we presented an improved version of the algorithm VF2, namely VF2 Plus. The effectiveness of the proposed improvements has been shown by comparing VF2 Plus with VF2 and other three state of the art algorithms. In particular, VF2 Plus has shown to be competitive with RI, the best algorithm in the state of the art on biological graphs. Moreover, VF2 Plus has gained more than one order of magnitude with respect to VF2 on the considered dataset. In the future works, we will strengthen the improvements proposed for VF2 Plus in order to obtain a renewed version of the algorithm.

References

1. Bonnici, V., Giugno, R., Pulvirenti, A., Shasha, D., Ferro, A.: A subgraph isomorphism algorithm and its application to biochemical data. BMC Bioinformatics 14 (2013)
2. Carletti, V., Foggia, P., Vento, M.: Performance Comparison of Five Exact Graph Matching Algorithms on Biological Databases. In: Petrosino, A., Maddalena, L., Pala, P. (eds.) ICIAP 2013. LNCS, vol. 8158, pp. 409–417. Springer, Heidelberg (2013)
3. Conte, D., Foggia, P., Sansone, C., Vento, M.: Thirty years of graph matching in Pattern Recognition. IJPRAI 18(3), 265–298 (2004)
4. Cordella, L., Foggia, P., Sansone, C., Vento, M.: A (sub)graph isomorphism algorithm for matching large graphs. IEEE Transactions on Pattern Analysis and Machine Intelligence 26, 1367–1372 (2004)

5. Dahm, N., Bunke, H., Caelli, T., Gao, Y.: Efficient subgraph matching using topological node feature constraints. Pattern Recognition (June 2014)
6. Foggia, P., Percannella, G., Vento, M.: Graph Matching And Learning In Pattern Recognition On The Last Ten Years. ... Journal of Pattern Recognition ... (2014)
7. Han, W.S., Lee, J.H., Lee, J.H.: Turbo Iso: Towards Ultrafast And Robust Subgraph Isomorphism Search In Large Graph Databases. In: ... of the 2013 International Conference on ..., pp. 337–348 (2013)
8. He, H., Singh, A.K.: Graphs-At-A-Time: Query Language And Access Methods For Graph Databases. In: Proceedings of the 2008 ACM SIGMOD International ..., pp. 405–417 (2008)
9. Huan, J., et al: Comparing graph representations of protein structure for mining family-specific residue-based packing motif. Journal of Computational Biology (2005)
10. Lacroix, V., Fernandez, C., Sagot, M.: Motif search in graphs: Application to metabolic networks. Transactions on computational biology and bioinformatics (Dicember 2006)
11. Larrosa, J., Valiente, G.: Constraint satisfaction algorithms for graph pattern matching. Mathematical Structures in Computer Science 12, 403–422 (2002)
12. McGregor, J.: Relational consistency algorithms and their application in finding subgraph and graph isomorphisms. Information Sciences 19(3), 229–250 (1979)
13. RCSB: Protein data bank web site (June 2015), http://www.rcsb.org/pdb
14. Shang, H., Zhang, Y., Lin, X., Yu, J.X.: Taming Verification Hardness: An Efficient Algorithm for Testing Subgraph Isomorphism, pp. 364–375 (2008)
15. Solnon, C.: Alldifferent-based filtering for subgraph isomorphism. Artificial Intelligence 174(12-13), 850–864 (2010)
16. Ullman, J.R.: An algorithm for subgraph isomorphism. J. Assoc. Comput. Mach. 23, 31–42 (1976)
17. Ullmann, J.: Bit-Vector Algorithms For Binary Constraint Satisfaction And Subgraph Isomorphism. Journal of Experimental Algorithmics (JEA) 15(1) (2010)
18. Vento, M.: A Long Tri. In: The Charming World Of Graphs For Pattern Recognition. Pattern Recognition, 1–11 (January 2014)
19. Vento, M., Jiang, X., Foggia, P.: International contest on pattern search in biological databases (June 2015), http://biograph2014.unisa.it
20. Zampelli, S., Deville, Y., Solnon, C.: Solving subgraph isomorphism problems with constraint programming. Constraints 15(3), 327–353 (2010)
21. Zhang, S., Li, S., Yang, J.: GADDI: Distance Index Based Subgraph Matching In Biological Networks. In: ... of the 12th International Conference on ... (2009)
22. Zhao, P., Han, J.: On Graph Query Optimization In Large Networks. Proceedings of the VLDB Endowment 3(1-2), 340–351 (2010)