On the variable ordering in the subgraph isomorphism algorithms

Vincenzo Bonnici, Rosalba Giugno

Abstract—Graphs are mathematical structures to model several biological data. Applications to analyze them require to apply solutions for the subgraph isomorphism problem, which is NP-complete. Here, we investigate the existing strategies to reduce the subgraph isomorphism algorithm running time with emphasis on the importance of the order with which the graphs vertices are taking into account during the search, called the variable ordering, and its incidence on the total running time of the algorithms. We focus on two recent solutions, which are based on an effective variable ordering strategy. We discuss their comparisons with the others algorithms present in the ICPR2014 contest on graph matching algorithms for pattern search in biological databases.

Index Terms—Biological graphs, subgraph isomorphism, variable ordering, search strategy, space search tree reduction.

1 Introduction

In research areas that deal with biological and chemical data, the advancements of the technology are yielding every day to the acquisition of more accurate data and interactions among them. Well-known examples are protein-protein interaction [1], [2], gene-regulatory [3], [4], [5], [6], drug-effect [7], [8], [9], [10], and disease correlation [11], [12], [13] networks. The mathematical structures used to model pairwise relations between data are the graphs. They consist of vertices to represent data and edges to connect them. Moreover, data such as molecules, drugs, proteins, and RNA structures have also two and three dimensional descriptions and are represented as graphs [14], [15].

Indeed, graphs serve as the underlying data structures for biomedical data repositories [16], [17], [18], [19], [20] and for the plenty of algorithms to analyze them [21], [22], [23], [24], [25]. For example, graph analysis allows finding network motifs [26], detecting active parts of molecules or regulatory circuits [27], [28], [29], computing networks, molecules and proteins similarity in base of the presence of a large amount of common subgraphs [30], [31].

The core procedure of the above algorithms implies solving the *subgraph isomorphism* (SubGI) problem. SubGI, sometimes called graph searching or graph matching, consists of finding the occurrences of a graph (called pattern) in another graph (called target). This is viewed as a generalization of the *graph isomorphism* (GI) problem, where pattern and target graphs have the same size. Recently, authors in [32] reported a polynomial time algorithm to solve GI, while SubGI is still considered belonging to the class of NP-Complete problems [33].

Solving SubGI is a complex task. Several techniques and heuristics are applied together to reach the goal with a reasonable running time. Often, the solution is a branch-and-bound algorithm over the the space of all possible matches, i.e.,*search space*, efficiently represented by a tree, *space search tree (SST)* [34].

The efficiency of a subgraph isomorphism algorithm mostly depends on the used heuristics to traverse the SST, which imply to apply constrains before and during the traversing for pruning unfair branches [35], [36], [37]. Some constraints can be more strength than others, and, usually, the level of strength is directly connected to the cost of applying them. A crucial aspect is to decide the *constraint ordering* to reduce the amount of verification in the pruning steps. Traversing strategy and constraint ordering do not affect the structure of the SST. Instead, the SST structure essentially depends on the order in which variables (pattern vertices) are taken into account, which is called *variable ordering*.

In this article, we review the characteristics of subgraph isomorphism algorithms present in literature, with particular emphasis on the classification and differentiation of the existing variable ordering strategies. Then, we discuss two algorithms, RI [38] and RI-DS [38], whose efficiency is due to the choice of suitable variable ordering with respect to apply extensive pruning rules. RI and RI-DS participated to the *ICPR2014 contest on graph matching algorithms for pattern search in biological databases* outperforming all other methods in terms of running time and memory consumption. The comparisons with reference to the competition are discussed in the result section.

2 Background

2.1 Basic Notions

A graph is a pair $G = (V, E)$, where V is the set of vertices, and $E \in V \times V$ is a set of edges connecting them. Graph vertices do not implicitly follow any order. However, an unique index is assigned to each of them and V is represented as an ordered set $\{v^1, v^2, \dots, v^{|V|}\}$.

If edges have directions, then G is denoted as a directed graph, otherwise G is an undirected graph. Given two vertices $v, v' \in V$, (v, v') indicates an undirected edge, and $\overrightarrow{(v, v')}$ is a directed edge going from the source vertex v to the destination vertex v'. The neighbourhood $N(v) = \{v' \in V : (v, v') \in E\}$ of a vertex v is the set of vertices connected to v. The neighborhood of a set of vertices $V' \subseteq V$ is $N(V') = \{\bigcup_{v \in V'} N(v) \bigcap V'\}$.

The degree of a vertex v is $|N(v)|$. In case of directed graphs, $N_{out}(v) = \{v' \in V : \overrightarrow{(v, v')} \in E\}$ is the outgoing neighborhood of v and $N_{in}(v) = \{v' \in V : \overrightarrow{(v', v)} \in E\}$ is its incoming neighborhood. A similarly distinction is made for out-degree and the in-degree of a vertex.

The pair (V, E) describes the topology of a graph. Labels may be associated to vertices and (or) edges. The notion of graph is extended to a quadruple $G = (V, E, \alpha, \beta)$, called labelled graph, where $\alpha : V \mapsto \Sigma_V$ and $\beta : E \mapsto \Sigma_E$ are two injective functions that map vertices and edges to their respective sets of labels Σ_V and Σ_E.

2.2 Problem Definition

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 *Graph Isomorphism problem* (GI) finds a bijective function $f : V_p \mapsto V_t$ that maps each vertex of G_p in G_t, and viceversa, by preserving the topology of both graphs and ensuring label compatibility. For a generic pattern vertex $v \in V_p$ and its mapped target vertex $f(v) \in V_t$, the label compatibility between them is ensured by the equivalence $\alpha_p(v) \equiv \alpha_t(f(v))$. Given two pattern vertices, $v, v' \in V_p$, the topology is preserved by a bijective correspondence between the pattern edges and the target edges. Formally, for each $(v, v') \in E_p$ than $(f(v), f(v')) \in E_t$, and for each $(v, v') \notin E_p$ than $(f(v), f(v')) \notin E_t$, as well as the edge labels compatibility $\beta_p((v, v')) \equiv \beta_t((f(v), f(v')))$ of exiting edges. The maps correspond to the matches or occurrences of the pattern graph in the target.

The *SubGraph Isomorphism problem* (SubGI) is defined by relaxing the conditions of the GI problem. Two kinds of SubGI problems exist. If f is injective than the GI problem becomes an *Induced Subgraph Isomorphism*. Whereas, by removing the constraint of non edge existence, the *Monomorphism* problem is obtained. The main difference between induced subgraph isomorphism and monomorphism is that the latter allows the existence of extra edges between the mapped vertices of the target graph, which do not exist in the pattern graph. In the following, we refer to SubGI problem without distinguish between induced and monomorphism unless necessary.

The subgraph isomorphism problem can be subcategorized in three ways [39]. *Testing*, which verifies the existence of at least one match of the pattern graph over the target graph; *Counting*, which calculates how many occurrences of the pattern graph are contained in the target graph; and *Listing*, which reports the exact locations of all such occurrences. Each one of such categories shows peculiar issues and can be solved with *ad-hoc* solutions. In this context we focus on the listing problem.

The SubGI problem can be also formalized in terms of a constraint satisfaction problem (CSP) [40]. Given a set of variables and a set of possible values, the problem is to find an assignment such that a set of constraints are satisfied. Constraints may involve multiple variables. In the case of SubGI, variables correspond to the pattern vertices V_p and values correspond to the target vertex set V_t. The label compatibility is modelled by unary constraints, while the topology is preserved by binary constraints representing the edges. At each variable is assigned a set of possible values, called *domain*, which changes during the searching phase [41]. Due to the strict connection between SubGI and CSP, in this article we may refer to pattern vertices as variables, pattern labels and pattern edges as constraints, and target vertices as values.

2.3 The Search Space

The function f of the SubGI problem definition is generally not unique, and the number of possible mappings equals the number of matches.

Let $\mu_p = (u_p^1, u_p^2, \ldots, u_p^{|V_p|})$ be a rearrangement of the ordering of the vertices in V_p, such that $u_p^i = v_p^j \in V_p$. The subgraph isomorphism problem can be seen as the combinatorial problem of finding all possible combinations of $|V_p|$ vertices of the target graph, $\mu_t = (u_t^1, u_t^2, \ldots, u_t^{|V_p|})$ such that $u_t^i = v_t^j$ for a $v_t^j \in V_t$. We represent a mapping by the ordered set of pairs $M = \{(u_p^1, u_t^1), (u_p^2, u_t^2), ..., (u_p^{|V_p|}, u_t^{|V_p|})\}$.

We denote with $M(i)$ the i-th pair of the mapping, $M(i) = (u_p^i, u_t^i)$; with $M[i]$ the first i-pairs of the mapping, $M[i] = \{(u_p^1, u_t^1), \ldots, (u_p^i, u_t^i)\}$; and with $M_p[i]$ the ordered set $(u_p^1, u_p^2, \ldots, u_p^i)$ (similarly for $M_t[i]$). We identify with $\mu_p[i]$ and $\mu_t[i]$ the first i elements of μ_p and μ_t, respectively. In Table1, we catalogued the used notation.

The set of all possible mapping is the *search space* of the problem, which is compactly represented as a tree, named the *Search Space Tree (SST)*, [34]. The tree has a dummy root for commodity, and two mappings M^1 and M^2 are branched together up to depth i iff $M^1[i] = M^2[i]$ and $M^1[i+1] \neq M^2[i+1]$. A branch from a root to a leaf of the SST encodes one of the possible mappings, whereas a branch from a root to an internal node of the SST represents a possible partial solution (or mapping). The order μ_p can be arbitrary chosen and it may vary among different mappings. For this reason, the SST for G_p and G_t may be not unique.

A simple brute force SubGI algorithm searches over all the possible mappings those which respect the SubGI constraints (see Section 2.2), which is obviously inefficient. Several methodologies are established in literature [35], [36], [38], [42]; alongside the CSP based ones [37], [43], [44], [45]. Those methodologies are a manifold of efficient strategies.

One aspect regards the selection of the SST, also referred as the SST construction. The main technique involved here regards the order, μ_p, in which variables are taken into account, called *variable ordering* (see Section 3.1). An other aspect is how the SST is visited. This process consists in traversing the SST from the root downward to the leafs following an opportune depth first search visit, such as partial solutions are verified and extended until a complete matching is found. More precisely, a basic strategy, called backtracking [41] (BT), visits the SST branches and verifies the SubGI constraints at each node of the tree. If the current matching pair does not satisfy the constraints, then all its unfair sub-branches are pruned and the algorithm backtracks to the previous partial solution. Improvements to the BT approach are given by the BM [46] (backmarking) and BJ [47] (backjumping) strategies. A SST may contain duplicate sub-trees which lead to redundant consistency checks. The aim is to avoid redundant checks by some straightforward bookkeeping for BM, or by detecting and avoiding parts of the SST that cannot contain any valid solution for BJ. An alternative to BT, and its variants, is to predict downward inconsistency by look-ahead procedures.

TABLE 1
Used notation

Symbol	Definition	Description
G	(V, E)	a generic graph
V	$\{v^1, v^2, \ldots, v^{\mid}V\mid\}$	the ordered vertex set of a graph
E	$\subseteq V \times V$	the set of edges of a graph
(v, v')	$(v, v') \in E$	an undirected graph edge
$\overrightarrow{(v, v')}$	$\overrightarrow{(v, v')} \in E$	an directed graph edge from u to v
$N(v)$	$\{v' \in V : (v, v') \in E\}$	the neighborhood of a vertex v
$N(V')$	$\bigcup_{v \in V'} \{N(v) \bigcap V'\}$	the neighborhood of a set of vertices $V' \subseteq V$
$N_{out}(v)$	$\{v' \in V : \overrightarrow{(v, v')} \in E\}$	the outgoing neighborhood of a vertex v
$N_{in}(v)$	$\{v' \in V : \overrightarrow{(v', v)} \in E\}$	the incoming neighborhood of a vertex v
G	(V, E, α, β)	a labeled graph
α	$\alpha : V \mapsto \Sigma_V$	a function which maps vertices to their labels
β	$\beta : E \mapsto \Sigma_E$	a function which maps edges to their labels
G_p	$(V_p, E_p, \alpha_p, \beta_p)$	a labeled pattern graph
G_t	$(V_t, E_t, \alpha_t, \beta_t)$	a labeled target graph
μ_p	$(u_p^1, u_p^2, \ldots, u_p^{\lvert V_p \rvert})$	a rearrangement of the ordering of the vertices in V_p
$\mu_p[i]$	$(u_p^1, u_p^2, \ldots, u_p^i)$	the first i elements of μ_p
μ_t	$(u_t^1, u_t^2, \ldots, u_t^{\lvert V_t \rvert})$	an order rearrangement of $\lvert V_p \rvert$ vertices of V_t
$\mu_t[i]$	$(u_t^1, u_t^2, \ldots, u_t^i)$	the first i elements of μ_t
(u_p^i, u_t^j)	for $u_p^i \in \mu_p, u_t^j \in \mu_t$	a mapping (or matching) of u_p^i to u_t^j, also called matching pair
M	$\{(u_p^1, u_t^1), (u_p^2, u_t^2), \ldots, (u_p^{\lvert V_p \rvert}, u_t^{\lvert V_p \rvert})\}$	a complete mapping by μ_p and μ_t
$M(i)$	(u_p^i, u_t^j)	the i-th pair of the mapping M
$M[i]$	$\{(u_p^1, u_t^1), (u_p^2, u_t^2), \ldots, (u_p^i, u_t^i)\}$	the first i pair of the mapping M
$M_p[i]$	$(u_p^1, u_p^2, \ldots, u_p^i)$	the first i pattern vertices of the mapping M
$M_t[i]$	$(u_t^1, u_t^2, \ldots, u_t^i)$	the first i target vertices of the mapping M
$M(u_t^i)$	$v_t^j \in V_t$, for $u_t^i \in M_t^s : u_t^i = v_t^j$	the mapped target vertex in M by invoking the original vertex order of V_t

These may involve the mere SubGI constraints or additional extra-constraints, usually less expensive to be evaluated, such as graph invariants, [48], [49], and properties related to the current partial matching, based on the neighborhood of a vertex [36], [42]. Graph invariants are also used to compute compatibility maps between pattern and target vertices, avoiding redundant computations [38], [45], [49].

In terms of CSP, the SubGI problem is solved by extending an assignment of a subset of variables until all variables are assigned, i.e., propagating the effects of a value assignment to the unassigned variable domains. If, after a propagation, one of the unassigned variable domain becomes inconsistent, then the current partial matching is discarded. For example, since a matching requires that all the mapped target vertices must be distinct, the assignment of a target to a pattern vertex is propagated by removing such target vertex from the domains of unmapped pattern vertices. If one of such domains becomes empty, then the current partial matching is discarded and the algorithm backtracks to the previous partial solution. In this direction, well known strategies are the forward-checking [50] (FC) and the arch-consistency [51], [52] (AC). FC separately propagates the effects of a selected value to all variables, like in the above example. AC forces arc consistency on all remaining variables by propagating edge constraints only to the neighborhood of the assigned pattern vertex or to further vertices. Finally, graph invariants are used to reduce the initial set of values of domains.

3 The Variable ordering strategy

In this section, we first categorize the variable ordering characteristics and then we describe the strategies used in literature (see Table2 for a summary view).

3.1 Variable ordering characteristics

3.1.1 *Static vs dynamic*

Variable ordering can be fixed before or during the traversing of SST. Let's refer to the first case as *static variable ordering* and to the second case as *dynamic variable ordering*. A dynamic variable ordering implies computations during the traversing in order to decide which variable will be the next in the branch. A static variable ordering defines the order once and it remains the same for all the branches.

3.1.2 *Pattern, target, state and domain driven*

A variable ordering may rely on properties of both pattern and target graphs, on the current state of the traversing, and on current values in variable domains. *Pattern* and *target driven variable orderings* take into account invariants owned by graphs, such as topology, degrees or label properties. *State driven variable ordering* is based on the current state of the computation, which also includes $M[i]$ (μ_p, μ_t or both), while *domain driven variable ordering* involves the values in the variable domains (i.e., their cardinality).

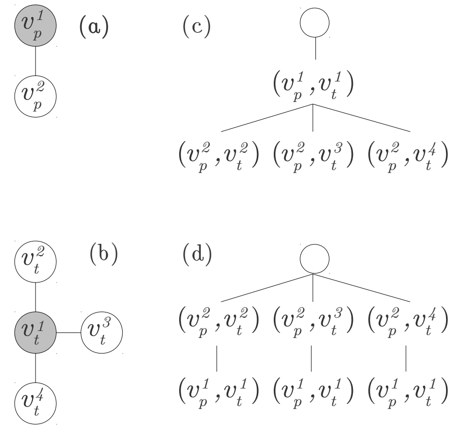
(a)
(b)
(c)
(d)
v_p^1
v_p^2
v_t^1
v_t^2
v_t^3
v_t^4
(v_p^1, v_t^1)
(v_p^2, v_t^2) (v_p^2, v_t^3) (v_p^2, v_t^4)
(v_p^1, v_t^1) (v_p^1, v_t^1) (v_p^1, v_t^1)

Fig. 1. (a) a pattern graph G_p and (b) a target graph G_t. Vertex labels are the colors white and gray. (c) is the resultant SST if the pattern vertex with the most unfrequent label is ordered first v_p^1. If v_p^2 is ordered before v_p^1 the SST in (d) is obtained.

3.1.3 Most-constraint and fail-first

The *most-constraint* principle enqueues in the ordering the variable that is subjected to the greatest number of constraints before the others. This principle generalizes to the *fail-first* principle where the constraints having an high probability of failing are put before in the ordering.

3.2 Variable ordering strategies in practice

3.2.1 Degree strategy

Highest-degree and *degree-1 ordering strategies* are pattern-driven, static and most-constraint variable ordering that order first the pattern vertex with the highest degree [53] or put the pattern vertices having degree equal to 1 in the last position of the ordering [54]. They generalize to the *degree strategy* in which pattern vertex are enqueued in descendant order according to their degree [35].

3.2.2 Unfrequently-feature and domain strategy

A strategy that applies the fail-first principle may choose the pattern vertex having the less frequent label in the target graph. This is a static and target-driven strategy, [50], which we call *unfrequently-feature* strategy.

The example in Figure 1 shows a pattern graph G_p having two nodes $\{v_p^1, v_p^2\}$ connected by one edge, for which $\alpha_p(v_p^1) \neq \alpha_p(v_p^2)$ and a target graph having the same label set of G_p, but only one vertex with the same label of v_p^1, identified by v_t^1. Searching first for v_p^1 may be more reasonable than searching for v_p^2, since the matching pair (v_p^1, v_t^1) is checked once instead of multiple times.

Note that, the domain of v_p^1 is smaller than the domain of v_p^2. Thus, algorithms which rely on domains reduction steps, implement the *singleton-domain* ordering strategy, which puts the variables with a domain cardinality equal to 1 in the first position of the ordering [37], [45], [50]. This is a static, domain and target-driven variable ordering.

Minimum-domain [50] strategy is a dynamic variable ordering based on domains, similarly to the one of the singleton-domain, but it works on domains having cardinality greater than 1. Given a partial matching $M[i]$, the next variable to be evaluated is the one with the smallest domain. By combining domain cardinalities and pattern vertex degrees, *degree-weighted-domains* is obtained [55], [56]. In [57], dynamic weights are assigned to domains. The domain weights are updated according to the number of times the constraints are violated. The variable with the minimum score is chosen. We refer to this as *weighted-domains* strategy. It is a dynamic, state-driven variable ordering based on the fail-first principle.

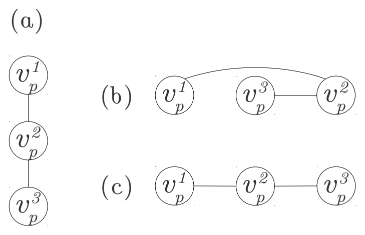

Fig. 2. (a) a pattern graph G_p and two of its possible variable orderings, (b) and (c). At $\mu_p[2]$, no edge constraints will be verified in (b), whereas in (c) the edge (v_p^1, v_p^2) is verifiable.

3.2.3 State-neighbourhood strategy

State-neighbourhood strategy is a fail-first, pattern, static and state variable ordering, which consists in choosing a pattern vertex that is a neighbour of the current partial order $\mu_p[i]$. Once $\mu_p[1]$ is chosen, $\mu_p[2]$ is selected over the set $N(\mu_p[1])$, and the process is repeated iteratively for each $\mu_p[i]$, until $i = |V_p| - 1$.

An explanation of this criterion can be shown by taking into account the unlabelled pattern graph G_p in Figure 2. If we choose the ordering $\mu_p = (v_p^1, v_p^3, v_p^2)$ (Figure 2 (b)), then all the topological constraints (the existence of the edges) can be verified only on the leaves of the SST's branches, namely at $\mu_p[i]$ for $i = |V_p| - 1$. As a consequence, several branches having $\mu_p[2] = (v_p^1, v_p^3)$ will be traversed without applying any constraints verification. Instead, if $\mu_p = (v_p^1, v_p^2, v_p^3)$ (Figure 2 (c)), then the constraints due to the edge (v_p^1, v_p^2) is verified at $i = 2$, and probably some of the branches of SST will be pruned.

A simple procedure to generate a state-neighbourhood strategy consists in choosing a pattern vertex v_p^i, putting it at the head of the variable ordering, and iteratively adding the pattern vertices as they are discovered by a breadth first visit starting from v_p^i [58].

The state-neighbourhood strategy can be also a dynamic, state, pattern and target-driven variable ordering as in [36], [42], which we call *State-neighbourhood*$^+$ strategy. Given a partial ordering $M_p[i]$, the next variable in the ordering is chosen in $N_{out}(M_p[i])$. If $N_{out}(M_p[i])$ and $N_{out}(M_t[i])$ are empty, then the next variable is chosen in $N_{in}(M_p[i])$. If $N_{in}(M_p[i])$ and $N_{in}(M_t[i])$ are empty, then the next variable is chosen in $\{V_p \setminus M_p[i]\}$. The reason why M_t is also taken into account, comes from the fact that the strategy is used to ensure the non-redundancy of the mappings.

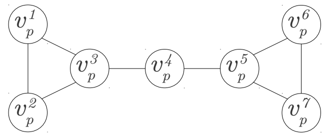

Fig. 3. A pattern graph where centrality measures may give an unfair variable ordering. Eccentricity [60] values of pattern vertices $\{v_p^1, v_p^2, \ldots, v_p^7\}$ are $\{0.25, 0.25, 0.33, 0.5, 0.33, 0.33, 0.25, \ 0.25\}$, respectively, and were calculated by the Cytoscape [61] pluging CentiScape [62]. Choosing v_p^4 as first vertex in the order goes against the most-constraint and fail first-principle and will yield an efficient strategy.

3.2.4 Maximum and sum -cardinality strategy

Maximum-cardinality strategy [59] provides an example of a state-neighbourhood strategy combined with the fail-first principle. Given a partial variable ordering $\mu_p[i]$, the next pattern vertex included in the ordering is the one with the highest number of neighbors in $\mu_p[i]$. The aim is to select the variable affected by the highest number of constraints that are verifiable at depth $i+1$ of the branching. In [45], the maximum-cardinality strategy is extended to cover the case of two variables that have the same number of neighbors in $\mu[i]$. The algorithm chooses the vertex v_p^k which maximizes the quantity $\sum\{|N(v_p^j)| : v_p^j \in \mu[i] \text{ and } v_p^j \in N(v_p^k)\}$. The variable indirectly affected by the highest number of already verified constraints is chosen. We refer to this ordering as the *sum-cardinality* strategy.

3.2.5 The choice of the first variable

The goodness of a state-neighbourhood ordering depends on the selection of the first variable. By selecting different starting vertices, the resultant variable orderings may consistently vary as well as their goodness to construct the SST. Generally, the vertex with the highest degree is selected to be the first [38], [59]. Alternatively, the vertex with the highest centrality or other graph centrality measurements could be selected. However, in a pattern graph like that shown in Figure 3, the most central vertex is v_p^4. Choosing such vertex goes against the most-constraint and fail first-principle and will yield an efficient strategy.

4 RI AND RI-DS METHODOLOGIES

In [38], the authors present RI, a methodology for solving the SubGI problem. It is based on a static, pattern and state-driven variable ordering. RI replaces costly predictive pruning verification (e.g., look-ahead procedures) with an effective variable ordering strategy.

4.1 RI Variable ordering strategy

Let $\mu[i]$ be a partial variable ordering. Let $v_p^k \in N(\mu[i])$ be a vertex candidate to be included in the order, its neighbors can be split in three sets (Figure 4):

1) $N_1(v_p^k) = \{v_p^j : v_p^j \in N(v_p^k) \text{ and } v_p^j \in \mu[i]\}$, those already included in the partial ordering;
2) $N_2(v_p^k) = \{v_p^j : v_p^j \in N(v_p^k) \text{ and } v_p^j \in \{N(\mu[i]) \ \mu[i]\}\}$, those who are not included but that are neighbors of at least one node in $\mu[i]$;

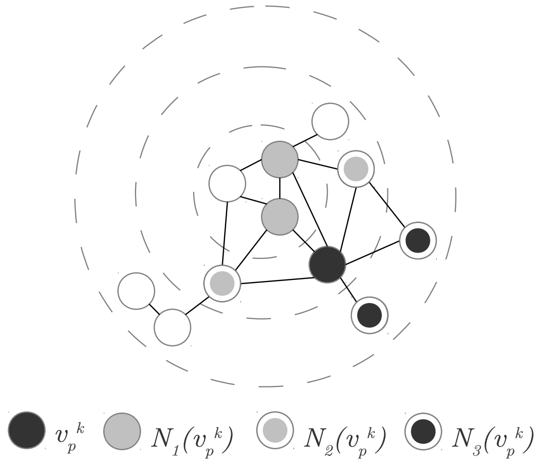

Fig. 4. A graphical representation of the 3-way neighborhood sets relatively to a vertex v_p^k and current partial ordering $\mu[i]$. Moreover, vertices in $\mu[i]$, $N(\mu[i])$ and the reminders are delimited by dashed circles from the most internal circle to the most external one, respectively.

3) $N_3(v_p^k) = \{v_p^j : v_p^j \in N(v_p^k) \text{ and } v_p^j \notin \{N_1(v_p^k) \cup N_2(v_p^k)\}\}$, all the remaining ones.

The variable ordering strategy of RI, namely 3-way-N, consists in comparing the cardinality of those three sets. Given two vertices $v_p^i, v_p^j \in N(\mu[i])$, v_p^i is preferred to v_p^j if $|N_1(v_p^i)| > |N_1(v_p^j)|$. If such cardinalities are the same, then v_p^i is preferred if $|N_2(v_p^i)| > |N_2(v_p^j)|$. If such second cardinalities are the same, then $|N_3(v_p^i)|$ and $|N_3(v_p^j)|$ are compared, choosing v_p^i if $|N_3(v_p^i)| > |N_3(v_p^j)|$. Finally, if such third cardinalities are the same, RI chooses randomly one among v_p^i and v_p^j. The aim is to choose the variables affected by the highest number of verifiable constraints, but at the same time maximizing the number of constraints that will become verifiable in the upcoming steps.

4.2 RI SST visit

The traversing procedure of the SST is a basic BT that verifies only the SubGI constraints. At step i of a partial matching, $M(i) = (v_p^k, v_t^j)$, RI checks the following constraints, in the order with which they are listed:

1) verify that v_t^j is not already included in the current partial matching, $v_t^k \notin \mu_t[i-1]$;
2) verify that $|N(v_p^k)| \le |N(v_t^j)|$;
3) verify vertex label compatibility, $\alpha_p(v_p^k) = \alpha_t(v_t^j)$;
4) verify the topology constraints, edges and their labels, for each $(v_p^k, v_p') \in E_p : v_p' \in \mu_p[i-1]$ than $(v_t^j, M(v_p')) \in E_t$ and $\beta_p((v_p^k, v_p')) = \beta_t((v_t^j, M(v_p')))$;
5) if induced sub-isomorphism is performed then verify that for each $(v_p^k, v_p') \notin E_p$ then $(v_p^j, M(v_p')) \notin E_t$.

The constraints 1), 2), 4), and 5) are the SubGI definition constraints. The constraint 3) is applied to prevent costly edge verification. In fact, the constraints ordering of RI reflects the case that the verification of the presence of target vertices in the current partial matching, is less costly

TABLE 2
Characteristics of the existing variable ordering strategies on solving SubGI problem.

Strategy	Static	Dynamic	Pattern	Target	State	Domains	Most-constraint	Fail-first
Degree	X		X				X	X
Unfrequently-feature	X			X				
Singleton-domain	X			X		X		
Minimum-domain		X			X	X		
Degree-weighted-domain	X	X	X		X	X	X	
Weighted-domain		X			X	X		X
State-neighbourhood	X		X		X			X
State-neighbourhood$^+$		X	X	X	X			
Maximum-cardinality	X		X		X		X	X
Sum-cardinality	X		X		X		X	X
3-way-N (RI)	X		X		X		X	X
3-way-N+D (RI-DS)	X		X	X	X	X	X	X

than label compatibility checks (this order can be tuned depending on label data complexity, use of compatibility maps, and devoted data structures).

4.3 RI-DS: Efficiency and memory consume trade off

RI has been developed to be both time and memory efficient. However, for NP-Complete problems solved by heuristic methodologies, efficiency and memory consumption is a typical trade-off subjected to the no-free-lunch principle [63], [64].

In [38], authors present RI-DS which differs from RI by precomputing a compatibility map among pattern and target vertices. This, in some applications, helps to reduce the total running time of the algorithm.

During the search phase, a matching pair (v_p^i, v_t^j) is probably tested more than once, i.e., it belongs to several matchings. The simple BT algorithm in RI verifies the label compatibility each time the pair (v_p^i, v_t^j) is taken into account. If the verification operation is relatively costly, the algorithm may suffer in terms of performance. On the contrary, RI-DS verifies the label compatibility just once and stores the result in a compatibility map C. In RI-DS, $C[i][j]$ is true if $\alpha_p(v_p^i) = \alpha_t(v_t^j)$ and $|N(v_p^i)| \leq |N(v_t^j)|$, false otherwise.

Moreover, before starting the searching process, RI-DS applies an AC reduction procedure to such initial domains (see Section 2). The RI-DS variable ordering strategy, namely 3-way-N+D, combines the RI's variable ordering with the singleton-domain strategy.

4.4 RI and RI-DS Data Structure

In RI and RI-DS, vertex labels are stored in a static array of size $|V|$. Given a vertex v^i, its adjacent list $Adj[i]$ is a static array of size $|N(v)|$. If the graph is ordered, only outgoing neighbors are stored. Since such data structures are static arrays, they are predisposed to be ordered and a binary search on them can be performed. Similarly to the adjacent list, given a vertex v^i, a static array stores the label of the edge $\overrightarrow{(v^i, Adj[i][j])}$. Finally, the compatibility map rows are stored using bit-vectors [35], [45], [65]. The choice of the above data structures are dictated by the aim to balancing the memory consumption with the running time of the algorithm.

5 RESULTS

5.1 Benchmarks

Biological graphs are very heterogeneous: (i) they can be sparse or dense, (ii) they can have labels on vertices and/or on edges or not, and (iii) they can vary in size from few to thousands of vertices.

In [66] and [67], the authors provided a set of target and pattern graphs independent of any applications. The dataset includes randomly connected graph, regular and irregular meshes (2D, 3D and 4D grids), and regular and irregular bounded valence (where the degree of vertices is bounded to a certain threshold) graphs. The target set is generated by varying some parameters of the graph structure [68]. The number of their vertices goes from 16 to 1296. Patterns were extracted from target graph by taking into account the ratio between their number of vertices.

In [38], a first general benchmark of biochemical and biological networks was presented. It includes chemical compounds, DNA, RNA and protein structures, contact maps of amino acids, protein-protein interaction (PPI) networks, and microbial networks. Patterns were extracted from targets, varying in size and density.

Inspired by the previous one, the dataset, used to evaluated the methodologies presented at the ICPR2014 contest of *Graph Matching Algorithms for Pattern Search in Biological Databases*, includes: biochemical compounds, protein structures and protein contact maps.

- The molecule dataset is formed by $10,000$ chemical compounds varying from 8 to 99 vertices and with an average degree of 2. The vertex label alphabet contains 5 distinct elements which represent the chemical elements (such as carbon, hydrogen, etc...) forming the compounds. A total of 50 patterns were extracted, with different number of vertices, $4, 8, 16, 32, 64$, ten for each size.
- The protein dataset contains 300 graphs having from 535 to $10,081$ vertices, and an average degree equal to 2. The vertex labels is still formed by the symbols

Fig. 5. Simulation results for the network.

of chemical elements, for a total of 5 distinct labels. 10 patterns were extracted for each size (number of vertices) $8, 16, 32, 64, 128, 256$, obtaining a total of 60 patterns.

- The contact maps dataset has 300 graphs having from 99 to 733 vertices and an average vertex degree equal to 20. The vertex label alphabet if formed by the 21 amino acid symbols. 60 patterns were extracted by varying the number of vertices $(8, 16, 32, 64, 128, 256)$.

Summarizing, the molecules dataset is formed by small and sparse graphs, the protein structure dataset contains large and sparse graphs, and the contact map one is a collection of medium size and dense graphs. All graphs have no labels on edges.

5.2 Comparisons

RI and RI-DS have been extensively compared in [38] on the monomorphism problem against VF2 [36], LAD [37] and FocusSearch [45]. Their performance have been shown in terms of running time, memory and visited size of the SST (i.e., the number of produced matching pairs during the search). They have also been compared in [69], on solving GI and induced SubGI. The performance of both algorithms were confirmed, except on solving GI over some instances of dense graphs. However, RI and RI-DS are optimized and designed to solve SubGI and not GI.

RI and RI-DS, together with other four algorithms participated to the *ICPR2014 contest on graph matching algorithms for pattern search in biological databases*.

- LAD [37] : a CSP method based on minimum-domain variable ordering which propagates the injectivity constraint (a target vertex cannot be mapped to two pattern vertices in the same solution).
- FC: a CSP based approach, referred here as FC, which combines two variants of forward checking [43] to propagate injectivity and edge constraints.
- L2G: a CSP based approach that combines a most-constraint variable ordering to the minimum-domain one, in a single dynamic variable ordering. It also applies a look-ahead procedure based on the vertex degree.
- PJ: a CSP based algorithm based on criteria presented in [70], [71] to deal with GI. Initial domains are computed by looking at label and degree compatibility, and an AC reduction procedure is applied. During the searching process, the algorithm propagates constraints up to a given distance (along the pattern graph) from the current variable.

All the presented approaches are developed in C++, except PJ that is developed in OCaml.

The contest comparisons required solving the induced SubGI in two cases: Finding all the occurrences of pattern graphs (Listing) or finding just the first one (Testing).

Algorithms were evaluated in terms of time and memory requirements. The results confirmed that RI and RI-DS outperform in average the other approaches. The graphics report the results provided by the context on the case of searching all occurrences (the methods showed similar behavior for the Testing case). Moreover, graphics include the performance of VF2 [36], one of the state of the art of graph matching algorithms.

Concerning the running time, CPS based approaches are affected by the costs of propagating constraints, updating variable domains and checking for their consistency, which may lead to unfair partial mappings. CSP based approaches work better in small or dense graphs, where the proximity of vertices helps the effectiveness of constraint propagation. However, RI and RI-DS have shown that avoiding such techniques and focusing on a good static variable ordering, leads to improving performance in terms of time requirements for all of the proposed datasets.

Figure 6 shows the running times on the sparse graphs of the molecule dataset. For low target sizes (number of vertices), domain based techniques can apply constraints efficiently, but they lose their efficiency on larger targets. On small sizes, CSP techniques are able to skim the initial domains. In fact, RI-DS shows similar trends to the CSP based approaches, even if it relies just on compatibility maps obtained by a single step of AC reduction. RI requires traversing the SST to obtain the same skimming level, which is generally more costly than compute compatibility maps.

Generally, RI outperforms the other methods on sparse graphs, as confirmed by the tests on the protein dataset, see Figure 7. Propagating constrains on large sparse graphs with respect to dense graphs does not improve the performance of CSP based approaches shown in [69] and [38]. However, as shown in Figure 8, RI-DS based on a simple initial domain reduction strategy outperforms all the other methods in dense graphs.

Concerning memory requirements, RI-DS outperformed all the other algorithms (not graphically shown here). All methods, with the exception of RI and RI-DS, are CSP based thus they need to dials with domains and their representation. Moreover, since they are based on propagation procedures, updated domains are stored by duplication of initial domains. Since, RI does not use domains, it is not affected by this problem. RI-DS uses domains just as compatibility map, it does not update them during the traversing phase and represents them by using bit-vectors. Moreover, all algorithms, except L2G, RI and RI-DS, use adjacent matrices to represent graphs, allowing a constant time for verifying edge existence but requiring quadratic space.

The mostly comparable approach with RI and RI-DS is L2G. It uses weak look-ahead procedures and it is based on choosing a good variable ordering. However, L2G relies on a dynamic variable ordering which needs additional cost respect to the static one of RI. This negatively affects the L2G's performance. Finally, the performance of RI, RI-DS and L2G could potentially be improved since they use adjacency lists instead of adjacency matrices, verifying edge existences with an asymptotic complexity bigger than $O(1)$.

6 Conclusion

In this article, we reviewed the general framework of subgraph isomorphism algorithms. We focused on the classification and differentiation of the variable ordering strategies.

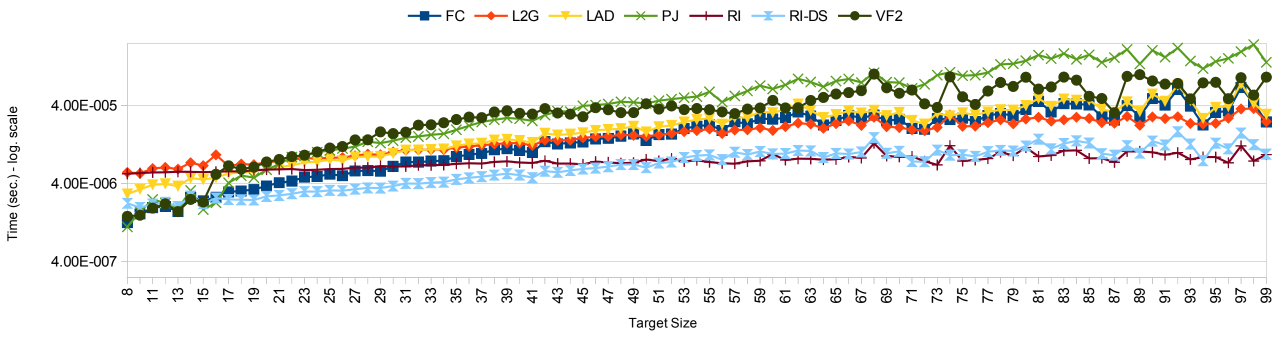

Fig. 6. Running times for finding all the occurrences over the molecules dataset.

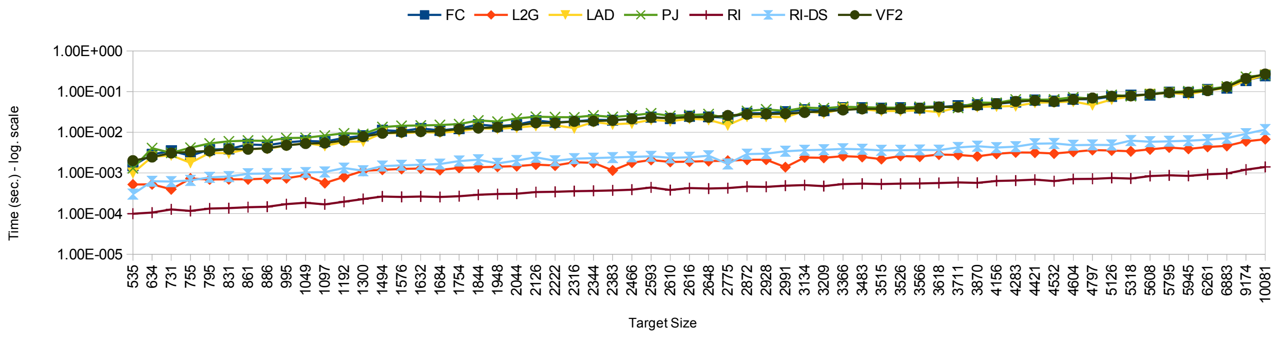

Fig. 7. Running times for finding all the pattern occurrences over the protein structure dataset.

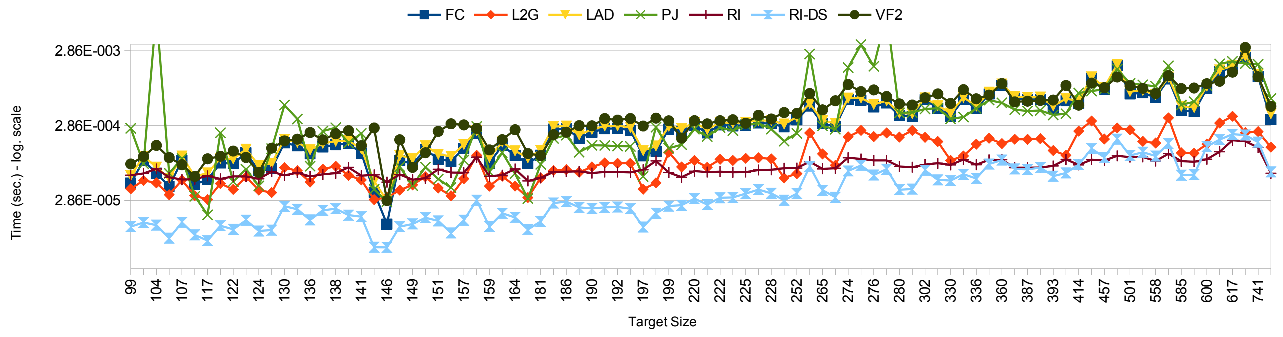

Fig. 8. Running times for finding all the pattern occurrences over the contact map dataset.

We showed the importance of variable ordering with respect to pruning rules. In order to do that, we described two algorithms RI and RI-DS and their performance in the ICPR2014 contest, where they outperformed all other methods.

References

[1] A. Mashaghi, A. Ramezanpour, and V. Karimipour, "Investigation of a protein complex network," *The European Physical Journal B-Condensed Matter and Complex Systems*, vol. 41, no. 1, pp. 113–121, 2004.

[2] S. Li, C. M. Armstrong, N. Bertin, H. Ge, S. Milstein, M. Boxem, P.-O. Vidalain, J.-D. J. Han, A. Chesneau, T. Hao *et al.*, "A map of the interactome network of the metazoan c. elegans," *Science*, vol. 303, no. 5657, pp. 540–543, 2004.

[3] P. Faccioli, P. Provero, C. Herrmann, A. Stanca, C. Morcia, and V. Terzi, "From single genes to co-expression networks: extracting knowledge from barley functional genomics," *Plant molecular biology*, vol. 58, no. 5, pp. 739–750, 2005.

[4] M. B. Gerstein, A. Kundaje, M. Hariharan, S. G. Landt, K.-K. Yan, C. Cheng, X. J. Mu, E. Khurana, J. Rozowsky, R. Alexander *et al.*, "Architecture of the human regulatory network derived from encode data," *Nature*, vol. 489, no. 7414, pp. 91–100, 2012.

[5] M. N. McCall, "Estimation of gene regulatory networks," *Journal of Postdoctoral Research*, vol. 1, no. 1, 2013.

[6] C. Christensen, J. Thakar, and R. Albert, "Systems-level insights into cellular regulation: inferring, analysing, and modelling intracellular networks," *IET systems biology*, vol. 1, no. 2, pp. 61–77, 2007.

[7] F. Iorio, R. Bosotti, E. Scacheri, V. Belcastro, P. Mithbaokar, R. Ferriero, L. Murino, R. Tagliaferri, N. Brunetti-Pierri, A. Isacchi *et al.*, "Discovery of drug mode of action and drug repositioning from

transcriptional responses," *Proceedings of the National Academy of Sciences*, vol. 107, no. 33, pp. 14 621–14 626, 2010.

[8] J. Lamb, "The connectivity map: a new tool for biomedical research," *Nature Reviews Cancer*, vol. 7, no. 1, pp. 54–60, 2007.

[9] M. Cokol, I. Iossifov, C. Weinreb, and A. Rzhetsky, "Emergent behavior of growing knowledge about molecular interactions," *Nature biotechnology*, vol. 23, no. 10, pp. 1243–1248, 2005.

[10] M. A. Yıldırım, K.-I. Goh, M. E. Cusick, A.-L. Barabási, and M. Vidal, "Drugtarget network," *Nature biotechnology*, vol. 25, no. 10, pp. 1119–1126, 2007.

[11] V. Janjić and N. Pržulj, "Biological function through network topology: a survey of the human diseasome," *Briefings in functional genomics*, p. els037, 2012.

[12] K.-I. Goh and I.-G. Choi, "Exploring the human diseasome: the human disease network," *Briefings in functional genomics*, p. els032, 2012.

[13] K. Wysocki and L. Ritter, "Diseasome an approach to understanding gene–disease interactions," *Annual review of nursing research*, vol. 29, no. 1, pp. 55–72, 2011.

[14] I. Daylight Chemical Information Systems. Daylight. [Online]. Available: http://http://www.daylight.com/

[15] Frowns. [Online]. Available: http://frowns.sourceforge.net/

[16] G. D. Bader, M. P. Cary, and C. Sander, "Pathguide: a pathway resource list," *Nucleic acids research*, vol. 34, no. suppl 1, pp. D504–D506, 2006.

[17] E. G. Cerami, B. E. Gross, E. Demir, I. Rodchenkov, Ö. Babur, N. Anwar, N. Schultz, G. D. Bader, and C. Sander, "Pathway commons, a web resource for biological pathway data," *Nucleic acids research*, vol. 39, no. suppl 1, pp. D685–D690, 2011.

[18] A. Chatr-aryamontri, B.-J. Breitkreutz, S. Heinicke, L. Boucher, A. Winter, C. Stark, J. Nixon, L. Ramage, N. Kolas, L. ODonnell *et al.*, "The biogrid interaction database: 2013 update," *Nucleic acids research*, vol. 41, no. D1, pp. D816–D823, 2013.

[19] V. Bonnici, F. Russo, N. Bombieri, A. Pulvirenti, and R. Giugno, "Comprehensive reconstruction and visualization of non-coding regulatory networks in human," *Frontiers in Bioengineering and Biotechnology*, vol. 2, p. 69, 2014.

[20] S. Turkarslan, E. J. Wurtmann, W.-J. Wu, N. Jiang, J. C. Bare, K. Foley, D. J. Reiss, P. Novichkov, and N. S. Baliga, "Network portal: a database for storage, analysis and visualization of biological networks," *Nucleic acids research*, vol. 42, no. D1, pp. D184–D190, 2014.

[21] A.-L. Barabasi and Z. N. Oltvai, "Network biology: understanding the cell's functional organization," *Nature Reviews Genetics*, vol. 5, no. 2, pp. 101–113, 2004.

[22] D. Yu, M. Kim, G. Xiao, and T. H. Hwang, "Review of biological network data and its applications," *Genomics & informatics*, vol. 11, no. 4, pp. 200–210, 2013.

[23] P. Csermely, T. Korcsmáros, H. J. Kiss, G. London, and R. Nussinov, "Structure and dynamics of molecular networks: A novel paradigm of drug discovery: A comprehensive review," *Pharmacology & therapeutics*, vol. 138, no. 3, pp. 333–408, 2013.

[24] A.-L. Barabási, N. Gulbahce, and J. Loscalzo, "Network medicine: a network-based approach to human disease," *Nature Reviews Genetics*, vol. 12, no. 1, pp. 56–68, 2011.

[25] A. Giuliani, S. Filippi, and M. Bertolaso, "Why network approach can promote a new way of thinking in biology," *Frontiers in genetics*, vol. 5, 2014.

[26] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon, "Network motifs: simple building blocks of complex networks," *Science*, vol. 298, no. 5594, pp. 824–827, 2002.

[27] N. Pržulj, "Biological network comparison using graphlet degree distribution," *Bioinformatics*, vol. 23, no. 2, pp. e177–e183, 2007.

[28] T. Milenkoviæ and N. Pržulj, "Uncovering biological network function via graphlet degree signatures," *Cancer informatics*, vol. 6, p. 257, 2008.

[29] S. Mangan and U. Alon, "Structure and function of the feed-forward loop network motif," *Proceedings of the National Academy of Sciences*, vol. 100, no. 21, pp. 11 980–11 985, 2003.

[30] A. P. Cootes, S. H. Muggleton, and M. J. Sternberg, "The identification of similarities between biological networks: application to the metabolome and interactome," *Journal of molecular biology*, vol. 369, no. 4, pp. 1126–1139, 2007.

[31] J. Lauri, "Subgraphs as a measure of similarity," in *Structural Analysis of Complex Networks*. Springer, 2011, pp. 319–334.

[32] S. Fankhauser, K. Riesen, H. Bunke, and P. Dickinson, "Suboptimal graph isomorphism using bipartite matching," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 26, no. 06, 2012.

[33] R. G. Michael and S. J. David, "Computers and intractability: a guide to the theory of np-completeness," *WH Freeman & Co., San Francisco*, 1979.

[34] N. J. Nilsson, *Principles of artificial intelligence*. Springer, 1982.

[35] J. R. Ullmann, "An algorithm for subgraph isomorphism," *Journal of the ACM (JACM)*, vol. 23, no. 1, pp. 31–42, 1976.

[36] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub) graph isomorphism algorithm for matching large graphs," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 26, no. 10, pp. 1367–1372, 2004.

[37] C. Solnon, "Alldifferent-based filtering for subgraph isomorphism," *Artificial Intelligence*, vol. 174, no. 12, pp. 850–864, 2010.

[38] V. Bonnici, R. Giugno, A. Pulvirenti, D. Shasha, and A. Ferro, "A subgraph isomorphism algorithm and its application to biochemical data," *BMC bioinformatics*, vol. 14, no. Suppl 7, p. S13, 2013.

[39] D. Eppstein, "Subgraph isomorphism in planar graphs and related problems," in *Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 1995, pp. 632–640.

[40] A. K. Mackworth, "Constraint satisfation," *Encyclopedia of Artificial Intelligence*, 1987.

[41] S. W. Golomb and L. D. Baumert, "Backtrack programming," *Journal of the ACM (JACM)*, vol. 12, no. 4, pp. 516–524, 1965.

[42] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "An improved algorithm for matching large graphs," in *3rd IAPR-TC15 workshop on graph-based representations in pattern recognition*, 2001, pp. 149–159.

[43] J. J. McGregor, "Relational consistency algorithms and their application in finding subgraph and graph isomorphisms," *Information Sciences*, vol. 19, no. 3, pp. 229–250, 1979.

[44] S. Zampelli, Y. Deville, and C. Solnon, "Solving subgraph isomorphism problems with constraint programming," *Constraints*, vol. 15, no. 3, pp. 327–353, 2010.

[45] J. R. Ullmann, "Bit-vector algorithms for binary constraint satisfaction and subgraph isomorphism," *Journal of Experimental Algorithmics (JEA)*, vol. 15, pp. 1–6, 2010.

[46] J. Gaschnig, "A general backtrack algorithm that eliminates most redundant tests." in *IJCAI*, 1977, p. 457.

[47] ——, "Experimental case studies of backtrack vs.{W} altz-type vs. new algorithms for satisficing assignment problems," 1978.

[48] B. D. McKay *et al.*, *Practical graph isomorphism*. Department of Computer Science, Vanderbilt University, 1981.

[49] N. Dahm, H. Bunke, T. Caelli, and Y. Gao, "Efficient subgraph matching using topological node feature constraints," *Pattern Recognition*, vol. 48, no. 2, pp. 317–330, 2015.

[50] R. M. Haralick and G. L. Elliott, "Increasing tree search efficiency for constraint satisfaction problems," *Artificial intelligence*, vol. 14, no. 3, pp. 263–313, 1980.

[51] A. K. Mackworth, "Consistency in networks of relations," *Artificial intelligence*, vol. 8, no. 1, pp. 99–118, 1977.

[52] F. Boussemart, F. Hemery, and C. Lecoutre, "Revision ordering heuristics for the constraint satisfaction problem," in *Proceedings of CPAI04 workshop held with CP04*, 2004, pp. 29–43.

[53] R. J. Wallace, "Factor analytic studies of csp heuristics," in *Principles and Practice of Constraint Programming-CP 2005*. Springer, 2005, pp. 712–726.

[54] D. Sabin and E. C. Ereuder, "Understanding and improving the mac algorithm," in *Principles and Practice of Constraint Programming-CP97*. Springer, 1997, pp. 167–181.

[55] C. Bessiere and J.-C. Régin, "Mac and combined heuristics: Two reasons to forsake fc (and cbj?) on hard problems," in *Principles and Practice of Constraint ProgrammingCP96*. Springer, 1996, pp. 61–75.

[56] B. M. Smith, "The brélaz heuristic and optimal static orderings," in *Principles and Practice of Constraint Programming–CP99*. Springer, 1999, pp. 405–418.

[57] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais, "Boosting systematic search by weighting constraints," in *ECAI*, vol. 16, 2004, p. 146.

[58] U. Čibej and J. Mihelič, "Search strategies for subgraph isomorphism algorithms," in *Applied Algorithms*. Springer, 2014, pp. 77–88.

[59] R. E. Tarjan and M. Yannakakis, "Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and se-

lectively reduce acyclic hypergraphs," *SIAM Journal on computing*, vol. 13, no. 3, pp. 566–579, 1984.

[60] O. Ore, *Theory of graphs*. American Mathematical Soc., 1965, vol. 38.

[61] P. Shannon, A. Markiel, O. Ozier, N. S. Baliga, J. T. Wang, D. Ramage, N. Amin, B. Schwikowski, and T. Ideker, "Cytoscape: a software environment for integrated models of biomolecular interaction networks," *Genome research*, vol. 13, no. 11, pp. 2498–2504, 2003.

[62] G. Scardoni, M. Petterlini, and C. Laudanna, "Analyzing biological network parameters with centiscape," *Bioinformatics*, vol. 25, no. 21, pp. 2857–2859, 2009.

[63] N. Biggs, *Algebraic graph theory*. Cambridge university press, 1993.

[64] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein *et al.*, *Introduction to algorithms*. MIT press Cambridge, 2001, vol. 2.

[65] C. Lecoutre, J. Vion *et al.*, "Enforcing arc consistency using bitwise operations," *Constraint Programming Letters (CPL)*, vol. 2, pp. 21–35, 2008.

[66] P. Foggia, C. Sansone, and M. Vento, "A database of graphs for isomorphism and sub-graph isomorphism benchmarking," in *Proc. of the 3rd IAPR TC-15 International Workshop on Graph-based Representations*. Citeseer, 2001, pp. 176–187.

[67] M. De Santo, P. Foggia, C. Sansone, and M. Vento, "A large database of graphs and its use for benchmarking graph isomorphism algorithms," *Pattern Recognition Letters*, vol. 24, no. 8, pp. 1067–1079, 2003.

[68] H. Bunke and M. Vento, "Benchmarking of graph matching algorithms," in *Proc. of the 2nd Workshop on Graph-based Representations*, 1999, pp. 109–114.

[69] V. Carletti, P. Foggia, and M. Vento, "Performance comparison of five exact graph matching algorithms on biological databases," in *New Trends in Image Analysis and Processing–ICIAP 2013*. Springer, 2013, pp. 409–417.

[70] L. Babai, P. Erdo s, and S. M. Selkow, "Random graph isomorphism," *SIAM Journal on Computing*, vol. 9, no. 3, pp. 628–635, 1980.

[71] P. Codenotti, "Distinguishing vertices of inhomogeneous random graphs," Technical Report IMA Preprint Series 2419, Institute for Mathematics and its Applications, University of Minnesota, Minneapolis, Minnesota, Tech. Rep., 2013.