SOFTWARE—PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2004; **34**:591–607 *Prepared using* speauth.cls *[Version: 2002/09/23 v2.2]*

Common subgraph isomorphism detection by backtracking search

Evgeny B. Krissinel* and Kim Henrick

European Bioinformatics Institute, Genome Campus, Hinxton, Cambridge CB10 1SD, UK

SUMMARY

Graph theory offers a convenient and highly attractive approach to various tasks of pattern recognition. Provided there is a graph representation of object in question (e.g. a chemical structure or protein fold), the recognition procedure is reduced to the problem of common subgraph isomorphism (CSI). Complexity of this problem shows combinatorial dependence on the size of input graphs, which in many practical cases makes the approach computationally intractable. Among the optimal algorithms for CSI, the leading place in practice belongs to algorithms based on maximal clique detection in the association graph. Bactracking algorithms for CSI, first developed 2 decades ago, are rarely used. We propose an improved bactracking algorithm for CSI, which differs from its predecessors by better search strategy and is therefore more efficient. We found that the new algorithm outperforms the traditional maximal clique approach by orders of magnitude in computational time.

KEY WORDS: Common subgraphs, backtracking algorithm

Introduction

Graph theory is a natural and convenient tool for dealing with objects that can be viewed as a connected set of more elementary subobjects. Such objects may be represented as graphs. In simple words, graph is a set of vertices connected by edges. This type of formalization is especially clear in the practically important case of molecular structures, however many different applications of graph theory are known (cf. Ref. [1] and other articles from that book). In the simplest case of chemical structures, atoms may be considered as graph vertices, and chemical bonds - as graph edges. Both atoms and bonds have properties, which may be

*Correspondence to: E. Krissinel, European Bioinformatics Institute, Genome Campus, Hinxton, Cambridge CB10 1SD, UK

assigned to the corresponding vertices and edges. Graphs, containing different types of vertices and/or edges, are called labeled. Graphs may be built on far more sophisticated objects, e.g. on secondary structure elements of a protein fold (vertices with vector properties) with edges measuring the distance between vertices and their mutual orientation (cf. Refs. [2, 3]).

Having constructed a graph representation of object in question, one immediately obtains a set of tools for measuring the similarity between same-type objects. This is achieved by graph matching, a procedure for identifying common subgraphs, or identical subsets of vertices in the compared graphs, connected by identical subsets of edges. The measure of similarity is then given by the size of maximal common subgraph (the one containing the maximal number of vertices) and, generally, by the properties of its vertices and edges. This technique plays an increasingly important role in many aspects of science and technology; the examples include protein-ligand docking [4], 3D structure recognition [2, 3, 5, 6, 7], modeling of active sites [8, 9], interpretation of molecular spectra [10, 11], prediction of biological activity [12], database searching [13, 14] and many others, including those outside chemo- and bioinformatics (e.g. computer vision and image recognition, cf. Refs. [15, 16, 17, 18, 19]).

Graph matching is known to be a computationally expensive procedure, which limits most of its applications. A number of graph-matching algorithms, both optimal and approximate, have been proposed over last 3 decades (see, e.g., reviews in Refs. [20, 21]). Optimal algorithms are those that guarantee the best solution(s) to be found, while approximate algorithms offer nearly-best solutions usually at considerably lower computation cost. Most of optimal algorithms for common subgraph isomorphism (CSI), used in various applications, are based on maximal clique detection in the association graph, as was first proposed in Refs. [22, 23]. In contrary, backtracking CSI algorithms (cf. Ref. [24]) are rarely used, presumably because of extra computational cost involved. It is widely accepted, however, that the problem of exact subgraph isomorphism (ESI, a special case of CSI when the maximal common subgraph coincides with one of the input graphs) is much more effectively solved by backtracking algorithm due to Ullman (UA, cf. Ref. [25]). This fact gives an implication that the backtracking approach may be efficient for CSI at least in cases that are *close* to ESI (we shall reveal the meaning of that further on in the paper).

In the next sections of the paper, we share our experience in constructing a backtracking CSI algorithm, which considerably outperforms the traditional maximal clique approach [22, 23]. The algorithm was initially derived from UA by modification of parts related to the expansion of partial solutions and rejection of unsuitable branches of the search tree. Our algorithm (hereafter referenced to as CSIA) differs from the backtracking CSI algorithm due to McGregor (cf. Ref. [24]) and has a controlling complexity parameter. We show that computational complexity of CSIA is bound by the same range as that of UA and which is considerably lower than complexity range shown by algorithms based on the maximal clique detection.

Definitions and the backtracking scheme

In what follows, we use the notion of *sets* (denoted by capital letters) of any objects (their *elements*, denoted by small letters). A set may be numbered, in which case its elements are indexed, e.g. $X = \{x_1, x_2, \ldots x_n\} = \{x_i\}_{i=1}^{n}$ (subscript i denotes the enumeration variable,

$n = |X|$ is number of elements in X). Sets are subject to direct multiplication: $Z = X \times Y = \{x_i y_j\}_{ij}$, overlaping: $Z = X \cap Y = \{x_i \equiv y_j\}_i$, addition: $Z = X + \{y\} = \{x_1, x_2, \ldots, y\}$ and subtraction: $Z = X - \{x_i\} = \{x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots\}$.

We shall represent graphs as 3-tuples $\mathcal{G} = [V, E, n]$, where $V = \{v_i\}$ and $E = \{e_{ij}\}$ are sets of vertices and edges, correspondingly, and $n = |V|$ is the number of vertices. Each element v_i and e_{ij} has a set of properties, which may be called as (composite or vector) labels, assigned to vertices and edges. We thus consider *labeled* graphs.

For the identification of matching vertices and edges, we introduce the vertex and edge comparison functions $\mu(v_i, v_j)$ and $\nu(e_{ij}, e_{kl})$, respectively. These functions return true if vertices or edges compare, and false otherwise. In most applications, these functions are comparing just two numbers, but generally they may allow for a certain threshold of discreapancy or more complicated rules applicable to composite lables, which is useful, e.g., for matching graphs representing fuzzy objects.

A subgraph of graph $\mathcal{G} = [V, E, n]$ is such graph $\mathcal{H} = [W, F, k]$ that $k \leq n$, $W \subseteq V$ and $F = E \cap (W \times W)$, or, in simple words, F is a subset W of k vertices from V, connected in graph $\mathcal{H}$ by *all and only* edges that connect these vertices in graph $\mathcal{G}$. The latter means, in particular, that if two vertices are not connected in graph $\mathcal{G}$, they can not be connected in its subgraph $\mathcal{H}$. Without loss in generality, we can consider that all unconnected vertices are connected by virtual *null* edges under the condition that *null* edges compare only with themselves. This formulation, which we shall use from here on, is more convenient for further discussion by eliminating the formal difference between connected and unconnected vertices (all graphs therefore appear as fully connected).

Common subgraphs $\mathcal{H}_1 = [V, E, k]$ and $\mathcal{H}_2 = [U, F, k]$ of two given graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ are those, of equal size k, that are isomorphic to each other. This means that there should be such numeration of subgraphs' vertices $x(i)$ and $y(i)$, that

$$\mu\left(v_{x(i)}, w_{y(i)}\right) = \mathsf{true} \quad \& \quad \nu\left(e_{x(i),x(j)}, f_{y(i),y(j)}\right) = \mathsf{true} \quad \forall\, i, j \in \{1 \ldots k\} \tag{1}$$

or, in simple words, all pairs of matching vertices in the subgraphs are connected by matching edges (including the virtual *null* edges). The problem of finding the common subgraphs can be now rephrased as looking for numbered sets $X = \{x(i)\}_{i=1}^{k}$ and $Y = \{y(i)\}_{i=1}^{k}$, satisfying conditions (1).

Backtracking scheme offers a simple and convenient approach to the solution (cf., e.g., Ref. [24]). Figure 1 presents a recursive version of the backtracking CSI algorithm (in first consideration, ignore the undefined "VMM $\mathcal{D}$, $\mathcal{D}_1$" in there). The algorithm essentially enumerates all possible mappings of vertices, which satisfy the conditions of subgraph isomorphism (1), in sets X and Y. Initially X and Y are set empty (cf. step 2 of the main program). On each entry to *Backtrack*, X and Y index vertices of partial common subgraphs found. The subgraphs may or may not be extended, which is checked by function *Extendable*. If solution is not extendable, it is output in step 17, and algorithm retreats. Otherwise, *Backtrack* picks a yet unmapped vertex v_i from graph $\mathcal{G}_1$ (step 2) and identifies set Z of all vertices from graph $\mathcal{G}_2$ (step 3) that may be mapped onto v_i without violation of subgraph isomorphism conditions (1) (candidate matchings). For each vertex w_j set Z, the algorithm extends the solution by appending X and Y with indices of v_i and w_j, respectively (cf. steps 5,6 in Figure 1). Then *Backtrack* is called recursively (step 8) in order to make further extensions.

Global graph $\mathcal{G}_1 = [V, E, n]$, $\mathcal{G}_2 = [W, F, m]$; set X, Y; integer n_{max}

1. **call** $Initialize(\mathcal{D})$
2. $X := \emptyset$, $Y = \emptyset$
3. $n_{max} := 0$
4. **call** $Backtrack(\mathcal{D})$
5. **stop**

procedure $Backtrack$ (VMM $\mathcal{D}$)

1. **if** $Extendable(\mathcal{D})$ **then**
2. $v_i := PickVertex(\mathcal{D})$
3. $Z := GetMappableVertices(v_i, \mathcal{D})$
4. **for** all $w_j \in Z$ **do**
5. $X := X + \{i\}$
6. $Y := Y + \{j\}$
7. $D_1 := Refine(\mathcal{D})$
8. **call** $Backtrack(\mathcal{D}_1)$
9. $X := X - \{i\}$
10. $Y := Y - \{j\}$
11. **done**
12. $V := V - \{v_i\}$
13. **call** $Backtrack(\mathcal{D})$
14. $V := V + \{v_i\}$
15. **else**
16. $n_{max} := \max(n_{max}, |X|)$
17. $Output(X, Y)$
18. **endif**

Figure 1. The backtracking scheme of the CSI algorithm. Capital letters denote numbered sets, and small letters - their elements, so that $X = \{x_i\}$. See details in the text.

After the recursive call, X and Y are restored (steps 9,10), which allows the algorithm to perform different mappings on each loopover. By going up and down the recursion, all possible extensions of common subgraphs, originating from the extension of subgraph of $\mathcal{G}_1$ by vertex v_i, are thus explored.

It is clear, however, that common subgraphs do not have to contain any particular vertex. Therefore, the search must be complemented by exploring all extensions of X that do not index the chosen vertex (v_i). For that purpose, in steps 12-14 v_i is temporary removed from *graph* $\mathcal{G}_1$, *Backtrack* is called recursively again, and after it returns, v_i is put back into $\mathcal{G}_1$.

In the most primitive implementation of this algorithm, $Extendable$ returns true if there are yet unmapped vertices in $\mathcal{G}_1$ and $\mathcal{G}_2$ left, $PickVertex$ returns just any unmapped vertex from

$V-\{v_{x(i)}\}_i$, $GetMappableVertices$ returns all vertices from $W-\{w_{y(i)}\}_i$, which, being mapped onto v_i and added to the solution, do not violate the conditions of subgraph isomorphism (1), $Initialize$ and $Refine$ do nothing. However, as practice shows, such implementation would not be the most efficient one.

It is obvious that efficiency of a recursive algorithm depends exponentially on the number of recursive calls it makes, while the number of operations, performed within a single call, makes the base of that exponent. As may be seen from Figure 1, the number of recursive calls may be decreased if $Extendable$ is able to determine whether the search leads to a *desirable* result on *further* recursion levels, and to return false if it does not.

Indeed, backtracking algorithms for graph matching use different techiques for terminating *undesirable* branches of the recursion tree. For example, Ullman algorithm [25] looks for exact subgraph isomorphism only, therefore any recursion branch not leading to common subgraph not coinciding with the lesser of the input graphs, is rejected as soon as it is identified as such. In the CSI algorithm due to McGregor [24], *undesirable* branches are defined as those not leading to subgraphs having more than maximal number of edges in common subgraphs already found.

In the next sections, we describe our strategy for pruning the recursion tree. Partly, the strategy is based on that employed by Ullman algorithm [25], however we make specific modifications for the CSI problem, and further improvement is done for the maximal CSI problem. We optimize the Ullman approach in checking for the conditions of subgraph isomorphism (1). In addition, we show that the choice of vertex v_i in step 2 of the algorithm in Figure 1 may have a pronounced effect on the performance. We also show that the time complexity of our algorithm is controlled by the size of common subgraphs it is looking for, and the larger is that size the *faster* is the search. Thanks to the improved strategy, our algorithm outperforms the traditional maximal clique technique for CSI problem, as well as UA when applied to ESI problem.

The CSI algorithm

In most applications, only sufficiently large common subgraphs are considered as a useful result of graph matching. For example, pattern recognition tasks normally require only the maximal common subgraphs (MCS) to be found, however if MCS is found but is insignificant in size, the recognition still yields negative. Without loss in generality or flexibility, we introduce a new parameter into CSI problem, namely the *minimal* size of common subgraphs to be found, n_0. Using this parameter, CSIA will reject branches of the recursion tree not leading to the sensible, from the application point of view, results, without spending time on finding them.

The central idea of UA [25] is based on using the vertex matching matrix (VMM) P. P_{ij} is true if vertex v_i of graph $\mathcal{G}_1$ can be matched to vertex w_j of graph $\mathcal{G}_2$, and false otherwise. Initially $P_{ij} = \mu(v_i, w_j)$, however, as the solution extends, P_{ij} takes into account that not all label-like vertices can be matched due to possible conflicts between edges connecting them (cf. conditions (1)). Using matrix P, UA concentrates only on those same-label vertices that do not have edge conflicts with vertices of *already* found subgraphs (cf. below). Naturally, in most cases the number of such candidate mappings decreases sharply, and P is getting very

sparse with the size of common subgraphs found. We therefore adopt a more efficient version of VMM, namely the matrix M, such that M_{ij} gives the *index* of vertex of graph $\mathcal{G}_2$ that is mappable onto vertex v_i of $\mathcal{G}_2$, $j = 1 \ldots L_i$. The 2-tuple $\mathcal{D} = [M, L]$ is referred as "VMM D" in Figure 1, and its intialization is demonstrated by procedure *Initialize* in Figure 2.

The compact form of VMM drastically simplifies the identification of mappable vertices in procedure *Backtrack*: our function *GetMappableVertices* merely returns all vertices of graph $\mathcal{G}_2$ indexed by ith row of matrix M (cf. Figure 2). Since VMM M includes only mappings that do not violate conditions (1) for already mapped vertices, extension of X with index i allows for extension of Y with any index $j = M_{ik}$, $k \in \{1 \ldots L_i\}$. However, once a particular extension extension is done, the VMM must be refined in order to exclude mappings that are not compatible with the *newly* mapped vertices v_i and w_j.

VMM is refined in step 7 of the bactracking scheme (cf. Figure 1). The refinement is done by function *Refine*, shown in Figure 2. The procedure is based on the comparison of edges, connecting the *last* mapped vertices ($v_{x(q)}$ and $w_{y(q)}$, where $q = |X| = |Y|$ is the size of common subgraphs) with all candidate mappings in both graphs. If edge between (unmapped) vertex v_i and $v_{x(q)}$ in graph $\mathcal{G}_1$ is not compatible with the edge between its mapping candidate $w_{M(i)(k)}$ and $w_{y(q)}$ in graph $\mathcal{G}_2$, the candidate is removed from the list (that is, removed from ith row of the VMM). It may be verified that checking only edges between candidate mappings and last mapped vertices is equivalent to checking the conditions of subgraph isomorphism (1) in full. Indeed, the initialization of VMM (procedure *Initialize*) guarantees that only vertices with compatible labels are listed as mappable, so we never check vertex labels in *Refine*. Then, the refinement is done at each extension of the solution, which guarantees that *Refine* receives a VMM, in which conditions (1) hold true for all edges between unmapped and mapped vertices, except only just mapped $v_{x(q)}$ and $w_{y(q)}$. It is therefore evident that using of VMM drastically reduces the number of comparison operations performed on each recursion level.

As was mentioned above, *Extendable* could always return true if there are unmapped vertices left in both graphs. Then each branch of the recursion tree would be explored to the very end of it. In order to terminate branches not leading to desirable solutions (cf. above) and thus save computational efforts, *Extendable* estimates the maximum possible size of common subgraphs that may be achieved by further continuation of the search. This is done on the basis of consideration that if, for yet unmapped vertex v_i from graph $\mathcal{G}_1$, the number of mapping candidates from graph $\mathcal{G}_2$ is zero, v_i will never be mapped (the mapping candidates are only removed as the search proceeds, cf. *Refine*). Therefore the maximal possible size of CSI s is given by the size of currently identified subgraphs $q = |X|$ plus number of non-empty rows in VMM $\mathcal{D} = [M, L]$ (i.e. those for which $L_i > 0$, see function *Extendable* in Figure 2). If s falls below n_0 (the minimal size of common subgarphs to be looked at) or is equal to q, *Extendable* returns false and the branch is abandoned. In the case of looking for maximal common subgraphs only, *Extendable* should also return false if s is less than the size of the largest common subgraphs found so far, n_{max} (n_{max} may be updated each time a common subgraph is output, cf. Figure 1, step 16). Then maximal common subgraphs will be found without finding *all* smaller subgraphs.

This procedure is illustrated by Figure 3. The Figure shows only one branch of the recursion tree starting with mapping (v_2, w_2), and includes only connected CSIs. On the very first level of recursion $r = 1$, CSIA removes vertex v_1 from graph $\mathcal{G}_1$ (v_1 is not mappable because there

Global integer n_0

procedure *Initialize* (VMM $\mathcal{D} = [M, L]$)

1. **for** all $v_i \in V$ **do**
2. $k := 0$
3. **for** all $w_j \in W$ **do**
4. **if** $\mu(v_i, w_j) =$ true **then**
5. $k := k + 1$, $M_{ik} := j$
6. **endif**
7. $L_i := k$
8. **done**

function *PickVertex* (VMM $\mathcal{D} = [M, L]$)
return v_i such that for all $v_i, v_j \in V - \{v_{x(k)}\}_k$, $0 < L_i \leq L_j$

function *GetMappableVertices* (vertex v_i, VMM $\mathcal{D} = [M, L]$)
return $\{w_{M_{ij}}\}_{j=1}^{L_i}$

function *Refine* (VMM $\mathcal{D} = [M, L]$)

1. VMM $\mathcal{D}_1 = [T, N]$
2. $q := |X|$
3. **for** all $v_i \in V - \{v_{x(i)}\}_{i=1}^{q}$ **do**
4. $l := 0$
5. **for** all $j \in \{M_{ik}\}_{k=1}^{L_i}$ **do**
6. **if** $\nu\left(e_{i,x(q)}, f_{j,y(q)}\right) =$ true **then**
7. $l := l + 1$
8. $T_{il} := j$
9. **endif**
10. $N_i := l$
11. **done**
12. **return** $\mathcal{D}_1$

function *Extendable* (VMM $\mathcal{D}_1 = [M, L]$)

1. $q := s := |X|$
2. **for** all $v_i \in V - \{v_{x(i)}\}_{i=1}^{q}$ **do**
3. **if** $L_i > 0$ **then** $s := s + 1$
4. **if** $s \geq \max(n_0, n_{max})$ and $s > q$ **return** true **else return** false

Figure 2. Procedures and functions used in the CSI algorithm shown in Figure 1. Global data from Figure 1 is used. See details in the text.

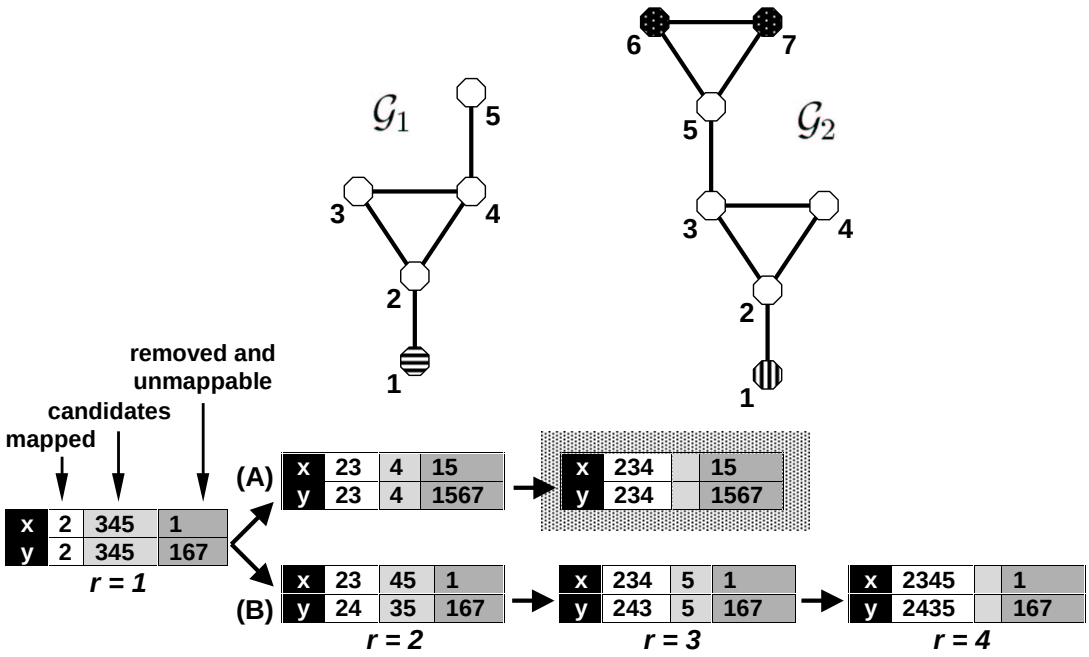

Figure 3. Example of the recursive CSI algorithm functionality. r stands for the recursion level, x and y list vertices of graphs $\mathcal{G}_1 = [V, E, n]$ and $\mathcal{G}_2 = [W, F, m]$, correspondingly. Mapped vertices are aligned vertically in "mapped" cells. Different filling of graph vertex symbols denotes different vertex labels. Only one branch of the recursion tree, starting from mapping (v_2, w_2), is shown, and only connected common subgraphs are considered. At minimal requested size of CSI $n_0 = 4$, the last mapping on the recursion level $r = 3$ in sub-branch A (highlighted by shaded background) is avoided by CSIA because the current size of graph $\mathcal{G}_1$ plus number of candidate matchings on level $r = 2$, sub-branch A, is less then n_0, see text for more details.

is no vertex in $\mathcal{G}_2$ with the same label). On the next recursion level, $r = 2$, CSIA explores two subbranches, A and B, which correspond to different mappings of vertex v_3. In subbranch A, mapping (v_3, w_3) makes vertex v_5 unmappable because of connectivity reasons, and the subbranch results in a CSI of size 3. Branch B leads to the largest CSI of size 4. If parameter n_0 is set to 4, then subbranch A can be terminated on recursion level $r = 2$, because the sum of mapped and potentially mappable vertices in graph $\mathcal{G}_1$ on that level is less than $n_0 = 4$. Thus, in this example, CSIA abandons subbranch A before detecting CSI of size 3 (highlighted in Figure 3 by shading). Although in this particular example the profit of avoiding level $r = 3$ of branch A does not look impressive, it is immense in most actual cases.

Finally, consider function $PickVertex$ (cf. Figure 2), which decides which vertex from graph $\mathcal{G}_1$ should be chosen for mapping on a particular level of recursion. Traditional algorithms simply pick vertices in the order of their numbering in the graph. This approach is optimal

only if input graphs are canonized and only for exact graph isomorphism problems (when two input graphs of equal size are checked for isomorphism). For exact subgraph isomorphism and common subgraph isomorphism problems, the approach gives no gain because subgraphs of a canonized graph do not have to be canonical. The backtracking scheme in Figure 1 suggests, however, that picking a vertex with *minimal* number of candidate mappings in $\mathcal{G}_2$ (given by set Z) accelerates the search in steps 4-11 due to lesser number of recusive calls the algorithm makes. Indeed, in the example in Figure 3, the algorithm may pick vertices v_3, v_4 or v_5 for mapping on level $r = 2$ (the Figure exemplifies picking of v_3). As may be seen, both v_3 and v_4 have two candidate mappings w_3 and w_4 from graph $\mathcal{G}_2$. Therefore, picking v_3 or v_4 results in branching (A and B in the Figure). However, v_5 has the only mapping candidate w_5 (no other same-labeled vertices are connected by *null* edge to already mapped v_2, w_2). Therefore, picking v_5 on level $r = 2$ leads straight to the solution without branching.

Because of removing vertices from $\mathcal{G}_1$ in step 12 of procedure *Backtrack* (cf. Figure 1), the algorithm may eventually come to mapping all remaining vertices on upper recursion levels (v_3 and v_4 in the above example). However, the corresponding to them branches of the recursion tree will be shorter due to a) lesser size of graph $\mathcal{G}_1$ (vertices removed) b) changing VMM, which may result only in decreasing the number of candidate mappings, cf. function $Refine$. Therefore, our version of $PickVertex$ searches for the minimal-length non-empty row i in VMM $\mathcal{D}$, and returns vertex v_i corresponding to that row (cf. $PickVertex$, Figure 2). Note that such vertex is always found, because function $Extendable$ terminates the search branch if no further mappings is possible.

Complexity analysis

Time complexity of graph-matching algorithms is estimated as the number of comparison operations (i.e. the number of calls to functions $\mu(v_i, w_j)$ and $\nu(e_{ij}, f_{kl})$) that they need to perform for obtaining the solution. Typically, that number depends drastically on the input graphs. Therefore, there are best-case and worst-case complexity estimates.

The best case of CSIA corresponds to the input of uniquely labeled graphs and $n_0 = \min(n, m)$. Then initialization of VMM, which takes $O(nm)$ comparisons, yields not more than one candidate mapping for each vertex of graph $\mathcal{G}_1$. The rest of comparison operations is done by function $Refine$, which makes not more than n per each call of *Backtrack*. Since $n_0 = \min(n, m)$ and all labels are unique, the branches of the recursion tree originating from call in step 13 are terminated immediately. This gives $O(n)$ calls of *Backtrack* with $O(n)$ comparison operations each. Considering the initialization step, the best-case complexity may be estimated as $O(nm)$.

In the case of unlabeled highly connected graphs, CSIA can not efficiently narrow the search no matter what the value of n_0 is. This represents the worst case for the algorithm. Here, *Backtrack* is called $O(m)$ times on each from $O(n)$ recursion levels, which gives an estimate of $O\left(m^n\right)$ calls total. During each call, function $Refine$ makes $O\left(mn\right)$ comparisons in order to filter out unacceptable mappings. Thus the algorithm complexity is bounded by

$$O\left(mn\right) \leq C_{CSIA} \leq O\left(m^{n+1}n\right) \tag{2}$$

Space complexity is estimated as the amount of RAM needed to keep the largest data structure used, that is VMM on each recursion level:

$$S_{CSIA} \approx O\left(mn^2\right) \tag{3}$$

Estimates (2) and (3) coincide exactly with those of Ullman algorithm [25]. This is expectable because CSIA represent essentially an advanced UA. A detail analysis of both algorithms shows that they have the same structure of nested loops, therefore the complexity limits are expected to coincide.

By comparison, time complexity of the traditional maximal clique approach (cf. Refs. [22, 23]) is bounded by

$$O\left(mn\right) \leq C_{LBKA} \leq O\left((nm)^n\right) \tag{4}$$

while its space complexity is given by the size of the association graph $\mathcal{G}_a = \mathcal{G}_1 \times \mathcal{G}_2$:

$$S_{LBKA} \approx O\left((mn)^2\right) \tag{5}$$

Comparison of estimates (2,3) and (4,5) implies that CSIA should outperform LBKA everywhere except the trivial best case, where both approaches are equally efficient. This is confirmed by our experience (cf. next Section), as well as by the widely accepted fact that UA considerably outperforms LBKA in exact subgraph isomorphism problem. Space complexity of both LBKA and UA/CSIA is not a limiting factor for modern computers.

Consider now the differences between UA and CSIA, which concern their time complexity, in more details. It may be verified that in the special case of $n_0 = \min(n, m)$ (ESI problem) CSIA reduces exactly to UA, and so does its compexity (in fact, CSIA is generally faster than UA in this case, see below). At $n_0 < \min(n, m)$, however, CSIA complexity shifts toward the upper estimate because a fewer number of branches of the search tree are terminated by function *Extendable*. The upper estimate in eq. (2) is most closely approached in the limiting case of $n_0 = 1$, when all branches are explored to the end (unless n_{max} is allowed to increase in step 16 of procedure *Backtrack*). It may be therefore concluded that CSIA is particularly efficient for finding larger subgraphs. In fact, the *close* is the situation to ESI (i.e. when the largest common subgraph is almost (except a few vertices) isomorphic to the smallest of $\mathcal{G}_1$ and $\mathcal{G}_2$), the *faster* the algorithm performs. Parameter n_0, the minimal size of common subgraphs to be found, therefore effectively controls complexity of the search. Using CSIA with wisely chosen value of n_0 (subject to particular application) allows to obtain CSI of sensible size much faster than by merely filtering out smaller subgraphs after size-unrestricted search.

In most practical cases, for properly coded algorithm, calculation of comparison functions $\mu\left(v_i, v_j\right)$ and $\nu\left(e_{ij}, e_{kl}\right)$ is as cheap as comparison of two integers. Then time complexity depends more on the *length* of internal loops in function *Refine* than on the actual number of comparisons. The compact form of VMM $\mathcal{D}$ in CSIA limits the inner loop in *Refine* to $O\left(\max_i(L_i)\right)$, while in UA this loop always runs from 1 to m in order to look up the whole row of uncompressed, although sparse, VMM. Repeating the above procedure for estimating the CSIA complexity, obtain

$$O\left(n \max_i(L_i)\right) \leq C_{CSIA} \leq O\left(\max_i(L_i)^{n+1} n\right) \tag{6}$$

Thus, if there are two kinds of vertices equally represented in the graphs, $\max_i(L_i) \approx m/2$ and the worst-case time expense may decrease by a factor of up to $\approx 2^{n+1}$, or 10^3 for 10-vertex graphs. For larger graphs, and if there are more labels in the graphs, the effect may be even stronger. In practice, we found this factor to be considerably less than predicted, because our estimates do not account for all accompanying operations. Still, we observed a 10 to 10^3-time increase in performance for graph having 50 and more vertices, once the compact form of VMM has been introduced.

Another considerable difference between CSIA and UA concerns the technique of picking vertices with minimal number of candidate matchings on each recursion level (cf. previous Section). It is difficult to provide an estimate for the effect of that modification. However, our experience suggests that the gain in performance may easily reach several orders of magnitude, in fact it is more noticeable than that of introducing the compact VMM.

Discussion

CSIA has been used for serving structural queries in the EBI-MSD ligand database and in the new EBI-MSD protein 3D comparison service SSM [26], as well as in the CCP4 Molecular Graphics Project [27]. The EBI-MSD ligand database contains chemical graphs of molecular structures (currently over 4000 entries), found as binding ligands in Protein Data Bank [28] depositions. The graphs include atoms (vertices) connected by chemical bonds (edges). Vertex labels are made of atom types and chirality indicator, edges are labeled such as to account for single, double, aromatic and triple bonds. Certain fragments, such as carbon atom with attached hydrogens, may be optionally represented as composite (pseudo-) atoms, which decreases the size of graphs and increases diversity of atom labels; both factors speed up the graph matching.

In order to get an idea about the actual time scale of graph matching, we run a series of all-to-all matches for all structures in the EBI-MSD ligand database, averaging the CPU time spent for structures of equal size. Figure 4 shows the averaged CPU time T_{UA} as obtained from the original Ullman algorithm [25] (ESI problem) as a function of the matched graphs' size n and m (measured on Pentium 4 1.2 Ghz PC). The surface $T_{UA}(n, m)$ is symmetrical in respect to n and m and ranges from ≈ 0.5 to 10 ms per match. Some non-monotonicity in the region of large graphs is explained by a relatively low fraction of larger structures in the database, which results in a poorer averaging. The picture suggests that the database represents a close to best-case complexity data set for UA. Indeed, the pronounced edge $n = m$, which dominates the surface, climbs up for approximately 2 orders of magnitude as n changes by 1 order of magnitude from 10 to 100 (for $n, m \leq 10$ the CPU measurements are less precise). As may be seen from eq. (2), that corresponds to the behaviour of the best-case complexity estimate $O(nm) \approx O(n^2)$. This conclusion agrees with the fact that normally graphs of chemical structures include vertices with several different labels and have low to medium connectivity (typically 1-3 edges per vertex).

Detection of CSI is a more laborious task than that of ESI. Figures 5 and 6 show the increase in the CPU time consumption as one proceeds from ESI to CSI. The surfaces were calculated using algorithm described in this paper. An attempt to use a maximal-clique algorithm (LBKA)

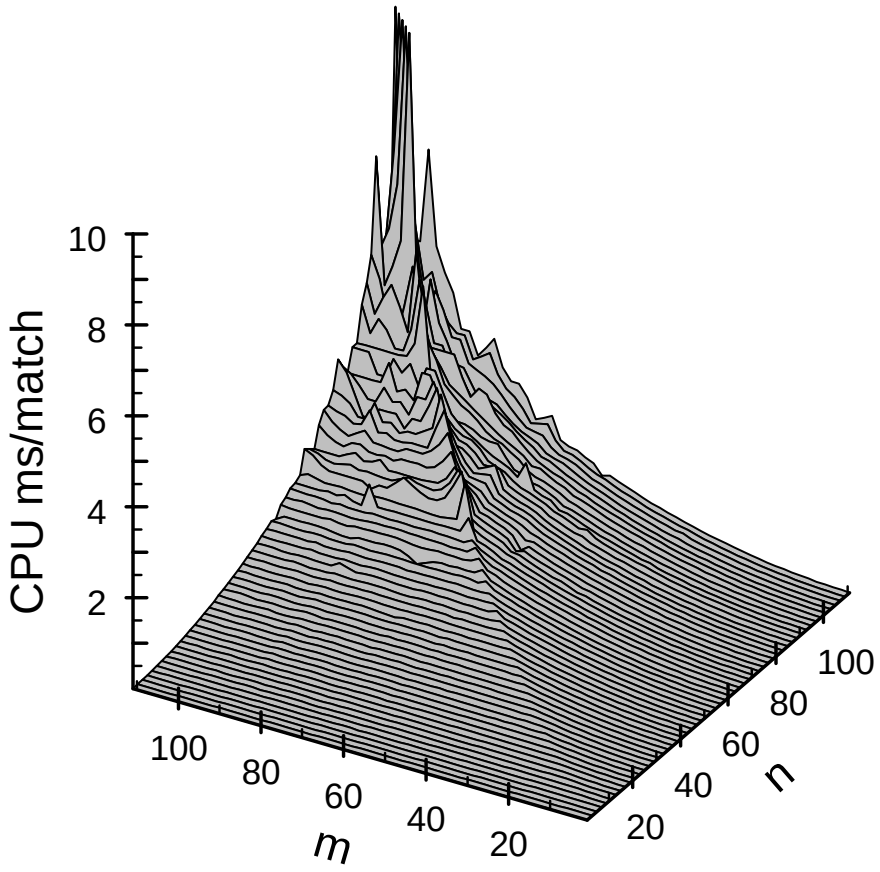

Figure 4. CPU time T_{UA} of ESI identification by the original Ullman algorithm [25] measured on all-against-all matches of chemical structures from the EBI-MSD ligand database [26] (over 4,000 entries), n and m stand for the size of input graphs.

did not allow to advance beyond graph size of 30 vertices without spending more than 50 hours of continuous calculations (after which the present study was actually initiated), therefore we do not compare with maximal-clique algorithms.

Figure 5 corresponds to the controlling complexity parameter $n_0 = \min(n, m) - 1$. This value of n_0 implies that CSIA neglects all common subgraphs that share all but more than 1 vertices in the smallest of matched graphs. One can imagine a naive application of UA to such CSI problem, which splits the smaller graph, say $\mathcal{G}_1$, into n_0 subgraphs $\mathcal{G}_1^i$ each having n_0 vertices, and then matches n_0 graph pairs $\{\mathcal{G}_1^i, \mathcal{G}_2\}$. On the close to best-case complexity data set, this would result in $\approx n_0$-time increase in the CPU time spent. Interestingly enough, such increase *is not* expected for the worst-case complexity data set. In order to demonstrate that,

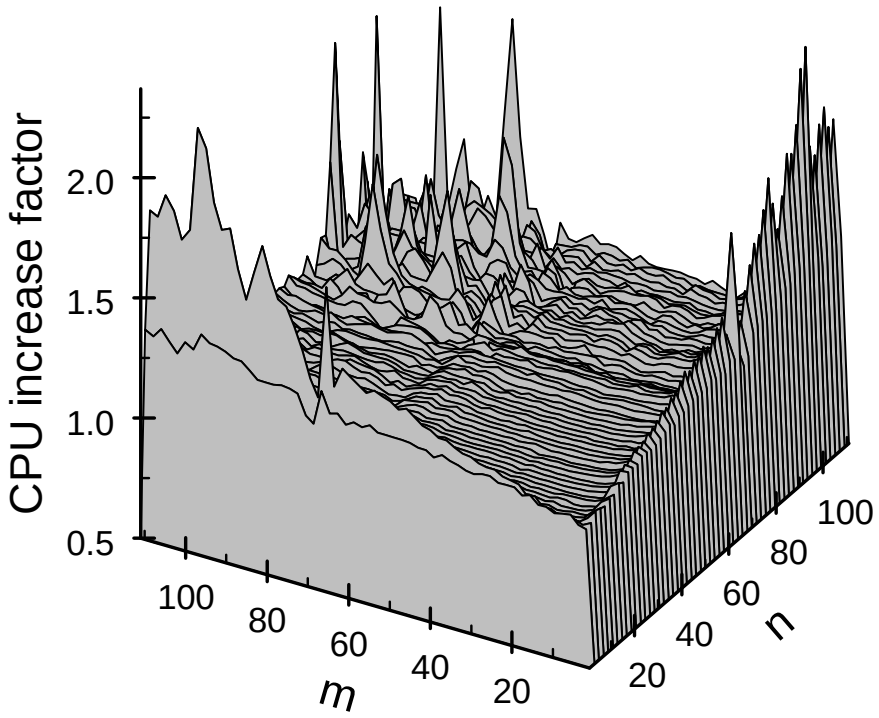

Figure 5. CPU time T_{CSIA} of CSI detection by the algorithm described in the present paper, in relation to T_{UA} from Figure 4. The controlling complexity parameter $n_0 = \min(n, m) - 1$. The database used is the same as that for Figure 4. The CSI problem is inherently harder than that of ESI, however the obtained increase factor is lower than may be expected, cf. discussion in the text.

consider again the edge $n = m$ of surface in Figures 4. The worst-case complexity estimate (cf. eq. (2)) gives there $O\left(n^{n+2}\right)$. Then time expense for matching n_0 graph pairs $\{\mathcal{G}_1^i, \mathcal{G}_2\}$ may be estimated as

$$O\left(n_0 n_0^{n_0+2}\right) = O\left(n_0^{n_0+3}\right) = O\left((n-1)^{n+2}\right) = O\left(n^{n+2}\right) \tag{7}$$

which may be generalized by induction on any n_0 as long as $n - n_0 \ll n$ and the algorithm works on the worst-case complexity data set. This result has a simple explanation: the worst-case complexity corresponds to the situations when function *Extendable* (cf. Figure 2) does not terminate branches of the search tree, and therefore search trees of graph pairs $\{\mathcal{G}_1^i, \mathcal{G}_2\}$ are nothing else but branches of the search tree of the original pair $\{\mathcal{G}_1, \mathcal{G}_2\}$. UA would merely go through all the same branches in either case.

As may be derived from Figure 5, for $n_0 = \min(n,m) - 1$ the CSIA strategy gives only up to $\approx$ 2-time increase in the CPU expense, as compared to ESI detection by UA, even for graphs as large as 110 vertices. Considering that the database used represents a close to best-case complexity data set for UA, one may conclude that CSIA performs much better than expected from the application of UA to branched subgraphs (cf. above). As was mentioned in the previous Section, although complexity of CSIA is formally bounded by the same limits as those of UA, it shifts toward the upper limit, approaching it most closely in the case of $n_0 = 1$. This is confirmed by general elevation of the surface in Figure 5 in the region of large n, m, however, occasional spikes, which are due to the specifics of particular graph pairs, suggest that the complexity shift is not a simple function of graph size. The sharp increase in the regions of $n \ll m$ and $m \ll n$ is due to the fact that in these regions $n_0 \ll \max(n,m)$ and therefore the number of potential CSIs to explore is significantly higher.

A detail exploration of the surface in Figure 5 yields that in some cases T_{CSIA}/T_{UA} falls below 1 (0.5 at minimum), which implies that CSIA may be *faster* than UA if used for ESI detection. This effect is solely attributed to the additional algorithmic optimizations described in previous Sections (compact VMM and choosing vertices with minimal number of candidate mappings). As found, switching those optimizations off makes $T_{CSIA} \geq T_{UA}$ in all cases.

The less n_0, the more laborious is the search for CSI. As explained in the previous Section, CSIA has to go through larger number of branches of the search tree when looking for smaller common subgraphs. In the branched subgraph approach (cf. above in this Section), the increase in computational expense may be estimated by a factor of $N_c = C_n^{n_0}$. Figure 6 demonstrates the ratio of CSIA computational time T_{CSIA} at $n_0 = \min(n,m) - 5$ to that of T_{UA} (shown in Figure 4). Detail investigation of the surface reveals that in vast majority of instances CSIA copes with matching at less than 10-time increase in computational efforts. Given the fact that at $n = 50$, $n_0 = 45$, where the above observation still holds, N_c reaches $\approx 10^6$ and thus CSIA performance may be rated as quite impressive. At the same time, particular graph pairs may require a considerable increase in CPU time, which in few cases is even larger than N_c. In Figure 6, the most relatively expensive matches (as compared to T_{UA}) are found in the region of $\max(n,m) \geq 60$ and $\min(n,m) \leq 20$; they are due to the low ratio of $n_0/\max(n,m)$, as explained for Figure 5 (cf. above). The region of large $n, m > 60$ shows only occasional moderate spikes. One possible explanation for these occasional spikes is that if a particular graph pair $\{\mathcal{G}_1, \mathcal{G}_2\}$ represents a close to best-case complexity data for ESI detection, it does not have to be so for the CSI task. For example, that may take place if the input graphs contain highly connected subgraphs made of vertices of the same type a, say $\mathcal{G}_1^a$ and $\mathcal{G}_2^a$. If, in addition to those subgraphs, $\mathcal{G}_1$ and $\mathcal{G}_2$ contain just a few vertices of different types b and c, respectively, UA will be able to immediately terminate all branches of the search tree and report negative on ESI (as there is no vertices b in $\mathcal{G}_2$ and vertices c in $\mathcal{G}_1$). In contrary to that, CSIA would eventually come to the stage when vertices b are removed from $\mathcal{G}_1$ and then fall into exploration of all possible mappings between highly connected $\mathcal{G}_1^a$ and $\mathcal{G}_2^a$ before giving an answer. If pair $\{\mathcal{G}_1^a, \mathcal{G}_2^a\}$ represents a worst-case complexity data for CSIA, T_{CSIA} may appear much larger than T_{UA}.

In the end of Discussion, let us stress the importance of algorithmic optimizations described in previous Sections: the compact form of VMM and choosing vertices with minimal number of candidate mappings. Without these optimizations (full-range VMM and choosing vertices

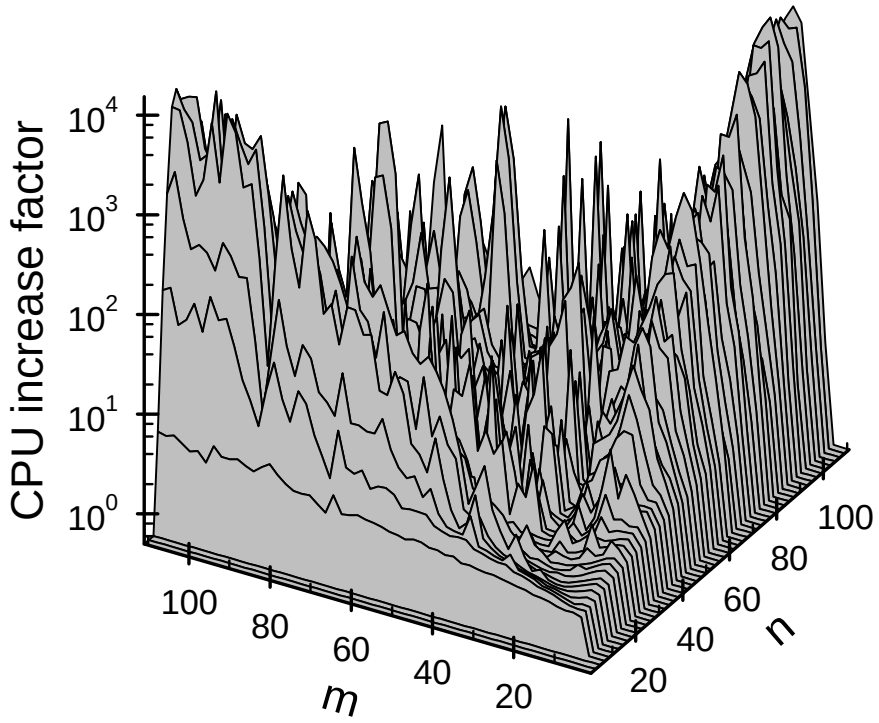

Figure 6. Same as Figure 5 but for controlling complexity parameter $n_0 = \min(n, m) - 5$.

in the order of their numbering in the graphs), calculation of surface in Figure 6 achieved only $n = m = 60$ vertices after 10 days of continuous work. Switching the modifications on allowed to make all the job in under 48 hours. We found that choosing vertices with minimal number of candidate mappings has a major impact for large graphs, while the effect of compact VMM prevails in the case of smaller graphs.

Conclusion

In the present paper, we have described an efficient backtracking algorithm for common subgraph isomorphism detection. The algorithm was found to be considerably more efficient than algorithm based on maximal clique approach (LBKA, cf. Refs. [22, 23]), and traditional backtracking schemes [24, 25] thanks to the additional algorithmic modifications introduced into the latter. The algorithm may be viewed as an advancement of Ullman algorithm (UA) for exact subgraph isomorphism [25]. The new algorithm, CSIA, inherits space and time

complexity of UA, which is much lower than that of LBKA, traditionally employed in CSI-related applications.

An attractive and conceptually new feature of CSIA is the presence of a controlling complexity parameter n_0, which allows one to save computational efforts by not looking at common subgraphs having less than n_0 vertices. The higher is n_0, the *faster* is CSIA. We found this parameter to be extremely useful in practical applications, where remote similarity poses little interest, if any. We demonstrated CSIA efficiency on cross-matching structures from the EBI-MSD ligand database [26], where CSIA is now routinely used as a database tool for structure recognition and retrieval.

It should be noted here that much attention during last two decades has been paid to the graph matching algorithms based on maximal clique detection. The approach received an improved heuristics, which considerably improves its efficiency. Authors of Ref. [33] compared one such advanced maximal-clique algorithm due to Cordella *et. al.* (cf. Ref. [34]), with a modification of McGregor's backtracking algorithm [8]. The modification affected only the definition of common subgraph, no improvement in strategy or heuristics has been reported. As found, McGregor's algorithm is up to 100 times more efficient for graphs with low connectivity, however it may be 10^4 times slower than the advanced maximal-clique algorithm on highly-connected graphs. To our experience, optimization of data structure and improvement of search strategy, proposed in the present paper, boost the performance by $4-6$ orders of magnitude as compared with non-optimized backtracking scheme exemplified by McGregor's algorithm. This allows one to expect that CSIA may compete successfully with modern algorithms based on maximal-clique detection.

Recently, a very efficient method for CSI, based on the decision tree algorithm, has been developed by Shearer *et. al.* [19]. The method is based on a preliminary classification of database graphs, initially proposed by Messmer *et. al* [29, 30] for ESI problem. The reported time complexity is $O\left(2^n n^3\right)$, which is considerably lower than the worst-case complexity of CSIA and depends only on the size of the input graph. The preproccessing step of the algorithm makes it essentially database-oriented. The weak point of the algorithm, however, is its space complexity, which explodes exponentially with the number of differently-labeled vertices and edges in the graphs. The latter represents an actual limiting factor; by authors' conclusion (cf. Ref. [19]), their algorithm is best for large databases of (very) small graphs (less than 20 vertices). This feature makes the decision-tree algorithm hardly applicable in many instances, even for matching 2D graphs representing chemical structures. In graphs representing true 3D-structures, edges and vertices may have continuous properties, like distance between the vertices and vertex orientation. In all such cases the space complexity of the decision-tree algorithm makes it inapplicable. In contrary, space complexity of CSIA does not limit its application, does not depend on the number of vertex and edge labels, and the more is that number, the *faster* is CSIA.

As described in this paper, CSIA is an algorithm for CSI identification in a pair of input graphs, therefore its application to a database is based on a sequential matching the database graphs to the input one. While this problem is more elegantly addressed by the decision-tree and decomposition network (ESI only) algorithms (cf. Refs. [19, 31]), we shall note that the application of CSIA to large databases may be further enhanced by employing database

pre-screening techniques in order to filter out the candidates, unsuitability of which may be established without graph matching.

Acknowledgement

EK is grateful for support from the BBSRC Collaborative Computational Project No. 4 in Protein Crystallography [32].

REFERENCES

1. D. H. Rouvray and A. T. Balaban, Chemical applications of graph theory, *Applications of Graph Theory*, R. J. Wilson and L. W. Beineke (eds), Academic Press, 1979.
2. E. M. Mitchell, P. J. Artymiuk, D. W. Rice and P. Willett, 'Use of techniques derived from graph theory to compare secondary structure motifs in proteins'. *J. Mol. Biol.* **212**, 151-166 (1990).
3. H. M. Grindley, P. J. Artymiuk, D. W. Rice and P. Willett, 'Identification of tertiary structure resemblance in proteins using a maximal common subgraph isomorphism algorithm'. *J. Mol. Biol.* **229**, 707-721 (1993).
4. F. Kuhl, G. Crippen and D. Friesen, 'A Combinatorial Algorithm for Calculating Ligand Binding'. *J. Comput. Chem.* **5**, 24-34 (1984).
5. D. A. Thorner, P. Willett, P. M. Wright and R. Taylor, 'Similarity searching in files of three-dimensional chemical structures: representation and searching of molecular electrostatic potentials using field-graphs'. *J. Computer-Aided Mol. Design* **11**, 163-174 (1997).
6. I. J. Bruno, N. M. Kemp, P. J. Artymiuk and P. Willett, 'Representation and searching of carbohydrate structures using graph-theoretic techniques'. *Carbohydrate Research* **304**, 61-67 (1997).
7. E. J. Gardiner, P. Willett and P. J. Artymiuk, 'Graph-theoretic techniques for macromolecular docking'. *J. Chem. Inf. Comput. Sci.* **40**, 273-279 (2000).
8. J. McGregor and P. Willett, 'Use of a maximal common subgraph algorithm in the automatic identification of the ostensible bond changes occurring in chemical reactions'. *J. Chem. Inf. Comput. Sci.* **21**, 137-140 (1981).
9. M. Arita, 'Graph Modeling of Metabolism'. *J. Jap. Soc. Artific. Intell.* **15**, 703-710 (2000).
10. L. Chen and W. Robien, 'Application of the Maximal Common Substructure Algorithm to Automatic Interpretation of 13C-NMR Spectra'. *J. Chem. Inf. Comput. Sci.* **34**, 934-941 (1994).
11. M. Cone, R. Venkataraghavan and F. McLafferty, 'Computer-aided interpretation of mass spectra. 20. Molecular structure comparison program for the identification of maximal common substructures'. *J. Am. Chem. Soc.* **99**, 7668-7671 (1977).
12. E. Gifford, M. Johnson, D. Smith and C. Tsai, 'Structure-Reactivity Maps as a Tool for Visualizing Xenobiotic Structure-Reactivity Relationships'. *Network Science* **2**, 1-33 (1996).
13. J. Raymond and P. Willett, 'Effectiveness of graph-based and fingerprint-based similarity measures for virtual screening of 2D chemical structure databases '. *J. Comput.-Aided Mol. Des.* **16**, 59-71 (2002).
14. S. J. Edgar, J. D. Holliday and P. Willett, 'Effectiveness of retrieval in similarity searches of chemical databases: a review of performance measures'. *J. of Mol. Graphics and Modelling* **18**, 343-357 (2000).
15. R. Horaud and T. Skordas, 'Stereo Correspondence Through Feature Grouping and Maximal Cliques'. *IEEE Trans. Pattern Anal. Mach. Intell.* **11**, 1168-1180 (1989).
16. M. Pelillo, K. Siddiqi and S. W. Zucker, 'Matching Hierarchical Structures Using Association Graphs'. *IEEE Trans. Pattern Anal. Mach. Intell.* **21**, 1105-1120 (1999).
17. B. Yang, W. Snyder and G. Bilbro, 'Matching oversegmented 3D images to models using association graphs'. *Image and Vision Computing* **7(2)**, 135-143 (1989).
18. F. Pla and J. Merchant, 'Matching Feature Points in Image Sequences through a Region-Based Method'. *Comput. Vision Image understand.* **66**, 271-285 (1997).
19. K. Shearer, H. Bunke and S. Venkatesh, 'Video indexing and similarity retrieval by largest common subgraph detection using decision trees'. *Pattern Recognition* **34**, 1075-1091 (2001).
20. J. Raymond and P. Willett, 'Maximum common subgraph isomorphism algorithms for the matching of chemical structures'. *J. Comput.-Aided Mol. Des.* **16**, 521-533 (2002).

21. H. Bunke, Recent advances in structural pattern recognition with applications to visual form analysis, In: *Arcelli, C.; Cordella, L. and G. Sanniti di Baja (eds.) Visual Form 2001*, Springer Verlag, LNCS 2059, 2001.
22. G. Levi, 'A note on the derivation of maximal common subgraphs of two directed or undirected graphs'. *Calcolo* **9**, 341-354 (1972).
23. C. Bron and J. Kerbosch, 'Algorithm 457 – finding all cliques of an undirected graph'. *Commun. A.C.M.* **16**, 575-577 (1973).
24. J. McGregor, 'Backtrack search algorithms and the maximal common subgraph problem'. *Software Pract. Exper.* **12**, 23-34 (1982).
25. J. R. Ullman, 'An algorithm for subgraph isomorphism'. *J. Assoc. Comput. Mach.* **23(1)**, 31-42 (1976).
26. H. Boutselakis, D. Dimitropoulos, J. Fillon, A. Golovin, K. Henrick, A. Hussain, J. Ionides, M. John, P. A. Keller, E. Krissinel, P. McNeil, A. Naim, R. Newman, T. Oldfield, J. Pineda, A. Rachedi, J. Copeland, A. Sitnov, S. Sobhany, A. Suarez-Uruena, J. Swaminathan, M. Tagari, J. Tate, S. Tromm, S. Velankar and W. Vranken, 'E-MSD: the European Bioinformatics Institute Macromolecular Structure Database'. *Nucl. Acids Res.* **31(1)**, 458-462 (2003).
27. E. Potterton, S. McNicholas, E. Krissinel, K. Cowtan and M. Noble, 'The CCP4 molecular-graphics project'. *Acta Cryst.* **D58**, 1955-1957 (2002).
28. H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov and P. E. Bourne, 'The Protein Data Bank'. *Nucl. Acids Res.* **28**, 235-242 (2000).
29. B. T. Messmer and H. Bunke, 'A new algorithm for error-tolerant subgraph isomorphism detection'. *IEEE Trans. Pattern Anal. Mach. Intell.* **20**, 493-504 (1998).
30. B. T. Messmer and H. Bunke, 'Error-correcting graph isomorphism using decision trees'. *Int. J. Pattern Recognition Artif. Intell.* **12**, 721-742 (1998).
31. B. T. Messmer and H. Bunke, 'Efficient subgraph isomorphism detection: a decomposition approach'. *IEEE Trans. on Knowledge and Data Engineering* **12**, 307-323 (2000).
32. Collaborative Computational Project, Number 4., 'The CCP4 Suite: Programs for Protein Crystallography'. *Acta Cryst. D* **50**, 760-763 (1994).
33. H. Bunke, P. Foggia, C. Guidobaldi, C. Sansone, and M. Vento, A comparison of algorithms for maximal common subgraph on randomly connected graphs, *Lecture notes in Computer Science 2396*, 2002, 123-132.
34. L. P. Cordella, P. Foggia, C. Sansone and M. Vento, An improved algorithm for matching large graphs, In: *Proc. of the 3rd IAPR-TC-15 International Workshop on Graph-based Representations, Italy*, 2001, 149-159.