

**uses vertex labels, but no edge labels.**  
*editing to include cost of edge label substitution.*

### Comparing Stars: On Approximating Graph Edit Distance

Zeng , Tung , Wang , Feng , & Zhou

we exploit the upper and lower bounds of edit distance to improve the search performance by filtering out graphs that definitely will not be in the answer set and thus avoid the expensive graph edit distance computation.

AppFull: efficient for performing approximate full graph over graph databases using graph edit distance as the similarity measure

Given a query graph  $q$  and the edit distance threshold  $\omega$ ,  
 for each graph  $g$  in graph database  $\mathcal{D}$ ,  
 $L_m(g, q)$  is first used to filter  $g$  if  $L_m$  is greater than  $\omega$  (lines 2-4), because  $\lambda > \omega$  must hold in the case of  $L_m > \omega$ .  
 Subsequently, if  $\tau(g, q) \leq \omega$ , we know that the edit distance between  $g$  and  $q$  must be no greater than  $\omega$ , and  $q$  can therefore be reported as a result (lines 5-8). If  $g$  passes the above two tests, then  $\rho$  is exploited further. Like in the case of  $\tau$ , if  $\rho(g, q) \leq \omega$ ,  $\lambda(g, q)$  must be no greater than  $\omega$  and  $g$  can be output as a result (lines 9-12). Finally, if  $g$  passes all the above three tests, then  $\lambda(g, q)$  must be computed (lines 13-15). The order of the above three tests is quite significant because the costs of their computation are different. Among three of them, the computation of  $L_m$  is the most efficient, while the computation of  $\rho$  is the most expensive. Therefore, if  $g$  does not pass an earlier test, the rest of the expensive tests are avoided.

Distance	Complexity	Distance	Complexity
$L_m$	$\Theta(n^3)$	$\tau$	$\Theta(n^3)$
$\rho$	$O(n^6)$	$\lambda$	NP-Hard

### Algorithm 3 APPFULL - Approximate Full Graph Search

**Input:** A query graph  $q$  and a graph database  $\mathcal{D}$

**Input:** Distance threshold  $\omega$

**Output:** All graphs  $g$  in  $\mathcal{D}$  s.t.  $\lambda(g, q) \leq \omega$

```

1. for each graph  $g \in \mathcal{D}$  do
2.   if  $L_m(g, q) > \omega$  then
3.     continue;
4.   end if
5.   if  $\tau(g, q) \leq \omega$  then
6.     report  $g$  as a result;
7.     continue;
8.   end if
9.   if  $\rho(g, q) \leq \omega$  then
10.    report  $g$  as a result;
11.    continue;
12.   end if
13.   if  $\lambda(g, q) \leq \omega$  then
14.    report  $g$  as a result;
15.   end if
16. end for

```

$$L_m \leq \lambda \leq \rho \leq \tau$$

## Comparing Stars: On Approximating Graph Edit Distance

Zeng , Tung , Wang , Feng , & Zhou

### class diagram for the AppFull and AppSub algorithms

```

StarStructure
=====
// index of root; label of root
rootIdx : int ; rootLabel : int;
// the adjacent vertexes of root in V[i] treated as leaves. values are labels.
// vLabels is sorted by value.
vLabels : int[]
// root -> V[i] edge labels, sorted by the vLabel sorting
eLabels : int[]
// the original indexes in V[i] of vLabels before sorting
origVIndexes : int[]
-----
-StarStructure()
+StarStructure(int root, int[] vLabels, int[] eLabels)
// protected. node and edge labelling functions which can be overridden with
// specialization. default is to use graph values
labels(Graph g, int rootIndex) : int[]
// -- static methods --
// Lemma 4.1
+calculateEditDistance(StarStructure s1, StarStructure s2) : double
+calculateEditDistanceNoRelabelling(StarStructure s1, StarStructure s2) : double

//create edit distance matrix for S(g_1) to S(g_2) from
//StarStructure.calculateEditDistance
createDistanceMatrix(StarStructure[] sg1, StarStructure[] sg2) : double[][]

//create multiset of IVI star structures from graph g with IVI vertices
createStarStructureMultiset(Graph g) : StarStructure[]

createStarStructure(Graph g, int rootIndex) : StarStructure

// sort array a in place, and return an array of the original indexes also sorted
sort(int[] a) : int[]
    
```

*w/ node and edge labels, both  $\geq 0$*

LEMMA 4.1. (Star Edit Distance) Given two star structures  $s_1$  and  $s_2$ ,

$$\lambda(s_1, s_2) = T(r_1, r_2) + d(L_1, L_2)$$

**+ a term for edge labels**

where

$$T(r_1, r_2) = \begin{cases} 0 & \text{if } l(r_1) = l(r_2), \\ 1 & \text{otherwise.} \end{cases}$$

$$d(L_1, L_2) = ||L_1| - |L_2|| + M(L_1, L_2)$$

$$M(L_1, L_2) = \max\{|\Psi_{L_1}|, |\Psi_{L_2}|\} - |\Psi_{L_1} \cap \Psi_{L_2}|$$

$\Psi_L$  is the multiset of vertex labels in  $L$ .

modifying Star Edit Distance to include edge labels:

$\lambda(s_1, s_2) = T(r_1, r_2) + d(L_1, L_2)$

where  $T(r_1, r_2) = 0$  if  $\text{label}(r_1) = \text{label}(r_2)$ , else = 1

$d(L_1, L_2) = ||L_1| - |L_2|| + M(L_1, L_2)$

where  $\psi(L)$  is the multiset of vertex labels on  $L$

where  $M(L_1, L_2) = \max(|\psi(L_1)|, |\psi(L_2)|) - |\text{intersection of } \psi(L_1) \text{ with } \psi(L_2)|$

edge labels: node labelling difference does not use distance, so edge will not either.

will compare edge label values after they have been sorted by the vertex label sorting. a cost of 1 when edge labels are not equal, else the cost is 0.

call this function  $d(L_1E, L_2E)$  and modified Star Edit Distance:

**$\lambda(s_1, s_2) = T(r_1, r_2) + d(L_1, L_2) + d(L_1E, L_2E)$**

# Comparing Stars: On Approximating Graph Edit Distance

Zeng , Tung , Wang , Feng , & Zhou

Symbols	Description
$deg(v)$	$ \{u (u, v) \in E\} $ , the degree of $v$
$\delta(g)$	$\max_{v \in V(g)} deg(v)$
$\lambda(g_1, g_2)$	the edit distance between graphs $g_1$ and $g_2$
$L_m(g_1, g_2)$	the lower bound of $\lambda(g_1, g_2)$
$\tau(g_1, g_2)$	the suboptimal value of $\lambda(g_1, g_2)$
$\rho(g_1, g_2)$	the refined suboptimal value $\lambda(g_1, g_2)$

## class diagram for the AppFull and AppSub algorithms

```

ApproxGraphSearchZeng
=====
-----
//AppFull for performing approximate full graph search.
// implements Algorithm 3
// q is the query, db is the graph to search, w is the edit distance threshold
+approxFullGraphSearch(Graph q, List<Graph> db, double w) : List<Graph>

//AppSub for performing approx subgraph search.
//the subgraph isomorphism problem is a computational task in which
//two graphs G and H are given as input, and one must determine
//whether G contains a subgraph that is isomorphic to H
+approxSubGraphSearch(Graph g, Graph h) : Graph

//pg5, col1 and col2. presumably, the best assignments of s_1_i to s_2_j
// have a best approach: greedy ordered by distMatrix?
calculateMappingDistance(StarStructure[] sg1, StarStructure[] sg2) : double

refine(Graph g, Graph h, int[][] permutation) : int[], double

```

during normalization of  $S(g_1)$  to  $S(g_2)$ ,  $k$  vertices with the label  $\epsilon$  are inserted into  $g_2$ . this is considered vertex relabelling so it does not change the edit distance between  $g_1$  and  $g_2$ . normalization is performed before the balanced weighted bipartite matching.

Given two multisets of star structures  $S_1$  and  $S_2$  with the same cardinality, and assume  $P : S_1 \rightarrow S_2$  is a bijection. The distance  $\zeta$  between  $S_1$  and  $S_2$  is

$$\zeta(S_1, S_2) = \min_P \sum_{s_i \in S_1} \lambda(s_i, P(s_i))$$

equivalent to solving the assignment problem

where  $P$  is an  $n \times n$  permutation matrix that is an orthonormal matrix having the property  $P^*P^T = P^AT^*P = I$  where  $I$  is the identity matrix.

NOTE:  $S_1$  originally has the same number or more vertexes than  $S_2$  and if the later, then normalization is  $S_2$  gets added vertexes with labels  $\epsilon$  (graph notation  $\epsilon \rightarrow u$  is insertion of  $u$ , and  $u \rightarrow \epsilon$  is deletion of  $u$ )

**DEFINITION 4.3. (Mapping Distance)** The mapping distance  $\mu(g_1, g_2)$  between  $g_1$  and  $g_2$  is defined as:

$$\mu(g_1, g_2) = \zeta(S(g_1), S(g_2))$$

## Comparing Stars: On Approximating Graph Edit Distance

Zeng , Tung , Wang , Feng , & Zhou

needed for AppFull (the approx full graph search)

### Lower Bound of Graph Edit Distance

Based on Lemma 4.2,  $\mu$  provides a lower bound  $L_m$  of  $\lambda$ , i.e.,

$$\lambda(g_1, g_2) \geq \underline{L_m(g_1, g_2)} = \frac{\mu(g_1, g_2)}{\max\{4, [\max\{\delta(g_1), \delta(g_2)\} + 1]\}}$$

$\tau(g, h)$ : the suboptimal value of  $\lambda(g, h)$

Assuming that the output from the Hungarian algorithm in Section 4.2.1

$$\zeta(S_1, S_2) = \min_P \sum_{s_i \in S_1} \lambda(s_i, P(s_i)) \quad \text{equivalent to solving the assignment problem}$$

leads to a mapping  $P^*$  from  $V(g)$  to  $V(h)$ , we can simply use Equation 1 from Section 3.1

$$C(g, h, P) = \sum_{i=1}^n \sum_{j=1}^n C_{i,j} P_{i,j} + \frac{1}{2} \|A^g - P A^h P^T\|_1 \quad (1)$$

to compute  $C(g, h, P^*)$ , denoted as  $\tau(g, h)$ .

Refine is guaranteed to be terminated in  $2n + n^2$  steps

### Algorithm 2 REFINES(g, h, P)

**Input:** two graph structures  $g$  and  $h$

**Input:** a permutation matrix  $P$  of  $g$  and  $h$

**Output:** refined suboptimal distance of  $g$  and  $h$

```

1.  $dist \leftarrow C(g, h, P)$ ;
2.  $min \leftarrow dist$ ;
3. for any pair  $(u_i, u_j) \in V(g)$  do
4.   get  $P'$  based on  $u_i$  and  $u_j$ ;
5.   if  $min > C(g, h, P')$  then
6.      $min \leftarrow C(g, h, P')$ ;
7.      $P_{min} \leftarrow P'$ ;
8.   end if
9. end for
10. if  $min < dist$  then
11.    $min \leftarrow \text{REFINE}(g, h, P_{min})$ 
12. end if
13. return  $min$ ;

```

$\rho(g, h)$ : the refined suboptimal value  $\lambda(g, h)$

$P^*$  is calculated using REFINES( $g, h, P$ ).

Refine will only find a local optimal solution

$\rho(g, h) =$

$$\lambda(g, h) = \min_P C(g, h, P) \quad (2)$$

$\lambda(g, h)$ : the edit distance between graphs  $g$  and  $h$

$$\lambda(g, h) = \min_P C(g, h, P) \quad (2)$$

# Comparing Stars: On Approximating Graph Edit Distance

Zeng , Tung , Wang , Feng , & Zhou

needed for AppSub (the approx sub graph search)

Symbols	Description
$deg(v)$	$ \{u (u, v) \in E\} $ , the degree of $v$
$\delta(g)$	$\max_{v \in V(g)} deg(v)$
$\lambda(g_1, g_2)$	the edit distance between graphs $g_1$ and $g_2$
$L_m(g_1, g_2)$	the lower bound of $\lambda(g_1, g_2)$
$\tau(g_1, g_2)$	the suboptimal value of $\lambda(g_1, g_2)$
$\rho(g_1, g_2)$	the refined suboptimal value $\lambda(g_1, g_2)$

hold. Accordingly, we devise a filtering algorithm **APPSUB** to perform  $\theta$ -subgraph search, in which  $L'_m$  is used as a filter:  
if  $L'_m(g_1, g_2) > \mathcal{L} + 2\theta$ ,  $g_2$  can be safely filtered.

if  $g_1$  is subgraph isomorphic to  $g_2$ , no vertex relabelling will exists in the optimal alignment that make  $g_2$  reach  $g_1$ , so the edit distance between two star structures is therefore redefined as:

$$\lambda'(s_1, s_2) = T'(s_1, s_2) + d(L_1, L_2) \quad \text{where}$$

$$T'(s_1, s_2) = \begin{cases} 2 + |L_1| + |L_2| & \text{if } l(r_1) \neq l(r_2), \\ 0 & \text{otherwise.} \end{cases}$$

In [35], Yan et al. introduced Grafil for performing approximate subgraph search by allowing edge relaxations(no vertex relabelling).

Assuming that  $g_3$  is the maximal common subgraph between a query graph  $g_1$  and a data graph  $g_2$ ,  
the number of edge relaxations in Grafil is defined as  $|E_1| - |E_3|$ .

\*Note that, the definition of edge relaxation implicitly implies that no vertex relabelling is allowed.

This similarity measure, however, does not take the vertex mismatches into account.

We therefore introduce the following similarity measure based on the graph edit distance to overcome this weakness of edge relaxation.

A graph  $g_1$  is said to be  $\theta$ -subgraph isomorphic to  $g_2$  if there exists a graph  $g_3$  s.t.

$g_3 \sqsubseteq g_2$  (i.e.  **$g_3$  is a subset of  $g_2$** ) and  **$\lambda'(g_1, g_3) \leq \theta$** .

Given a query  $g_1$  and a graph database  $D$ , the problem of  $\theta$ -subgraph search is to find out all graphs  $g_2$  in  $D$  of which  $g_1$  is a  $\theta$ -subgraph.

**if  $g_1$  is a  $\theta$ -subgraph of  $g_2$ ,  $\lambda'(g_1, g_2) \leq \mathcal{L} + 2\theta$  must hold**

AppSub inherently supports both of two kinds of subgraph search, i.e., traditional subgraph search[37] and containment search[9]

## A Coding Method for Efficient Subgraph Querying on Vertex- and Edge-Labeled Graphs

• May 2014 PLoS ONE 9(5):e97178

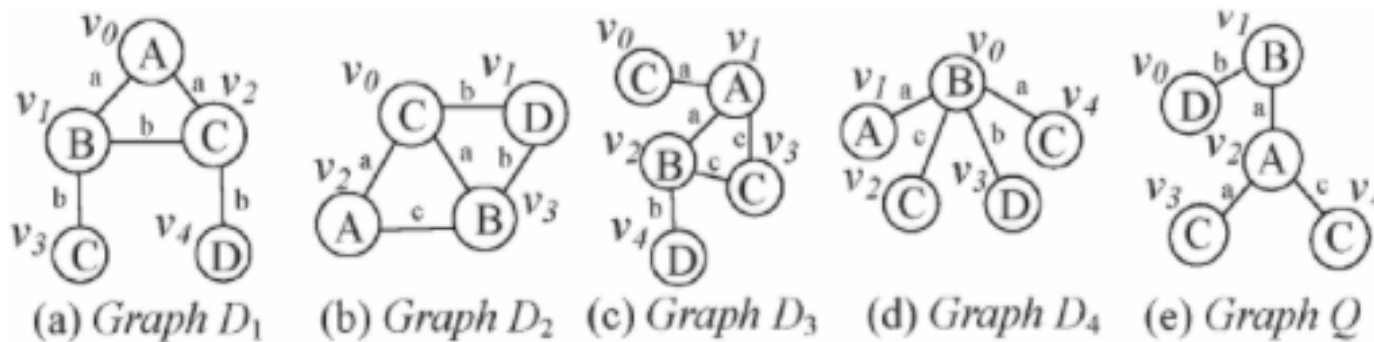
DOI:10.1371/journal.pone.0097178

Zhu et al.

The subgraph query problem is to retrieve all the supergraphs of a given graph from a graph database. It can be defined as follows: for a large graph database  $D = \{D_1, D_2, \dots, D_n\}$  and a query graph  $Q$ , subgraph query is to find all the graphs  $D_i$  ( $i=1,2,\dots,m < n$ ) such that  $Q$  is a subgraph of  $D_i$ .

Fig. 1 shows an example of subgraph query, where the graph database consists of graphs  $D_1, D_2, D_3$  and  $D_4$ , and  $Q$  is the query graph. Obviously, only graph  $D_3$  contains  $Q$ .

example  
for tests



**Figure 1. An Example of Subgraph Query.** Four labeled graphs (a) *Graph D<sub>1</sub>*, (b) *Graph D<sub>2</sub>*, (c) *Graph D<sub>3</sub>*, and (d) *Graph D<sub>4</sub>* compose the database, and (e) *Graph Q* is a query graph.

doi:10.1371/journal.pone.0097178.g001

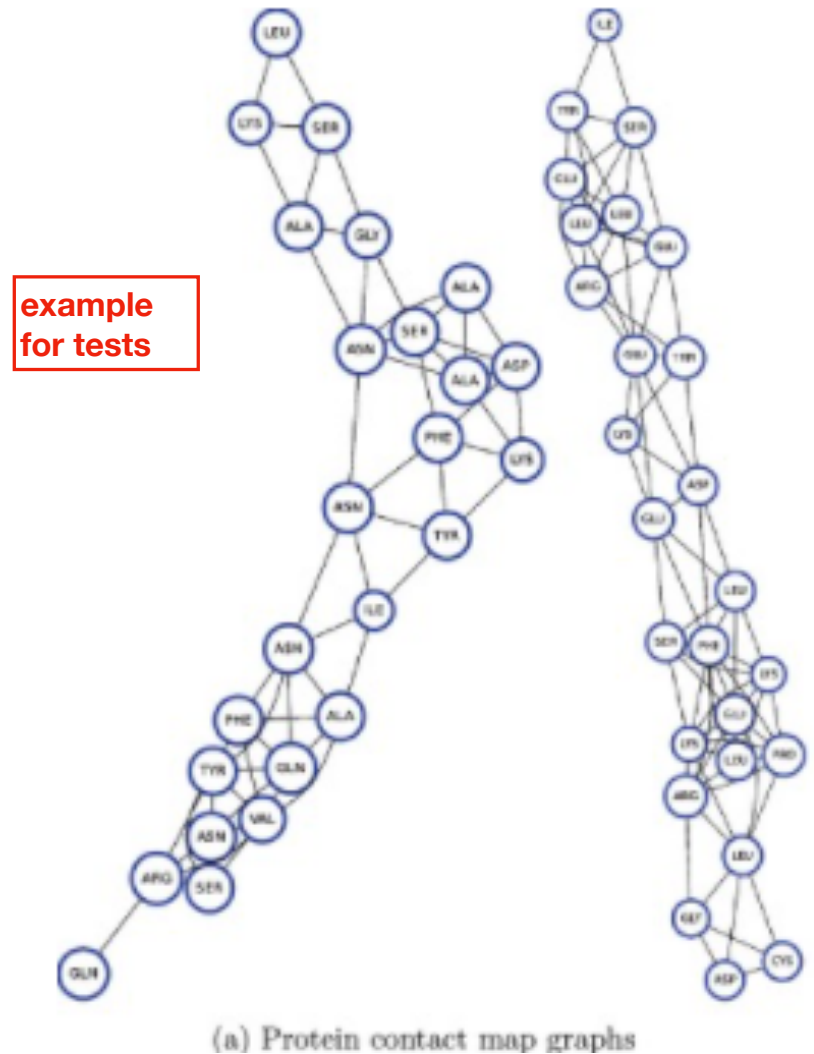
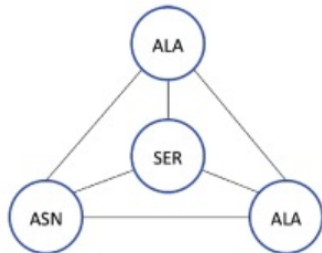
Dipali Pal, Praveen Rao, Vasil Slavov, Anas Katib,  
Journal of Computer and System Sciences, Volume 82, Issue 6, 2016, Pages 1112-1143,  
<https://doi.org/10.1016/j.jcss.2016.04.002>.

Given a query graph  $Q$ , an exact subgraph matching query finds all graphs in  $D$  that contain a subgraph that is isomorphic to  $Q$ .  
(This query is also referred to as a subgraph containment query in prior work.)

Given a query graph  $Q$  and a distance threshold  $d$ , an approximate graph matching query finds all graphs in  $D$  whose edit distance with  $Q$  is at most  $d$ .

An edge exists between two residues if the distance between their c<sub>α</sub> atoms is below a particular threshold [14].

an exact subgraph matching query on the protein contact map graphs in Fig. 1(a) by ALA, SER, ASN, and ALA: only the data graph on the left-hand side of Fig. 1(a) contains a subgraph isomorphic to the query.





**Fast processing of graph queries on a large database of small and medium-sized data graphs,**

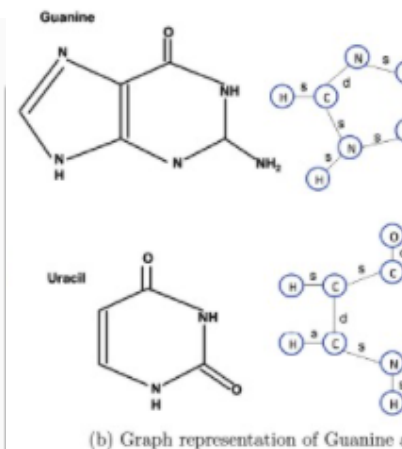
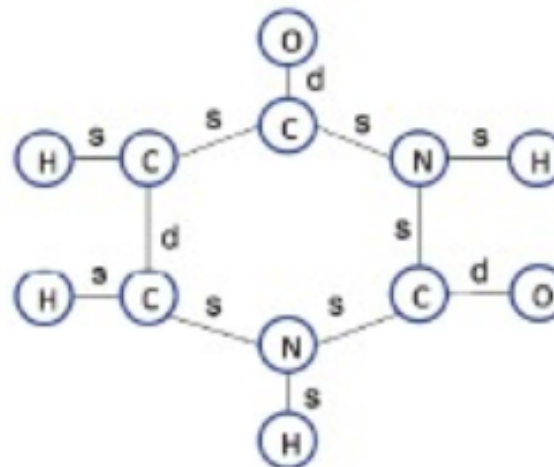
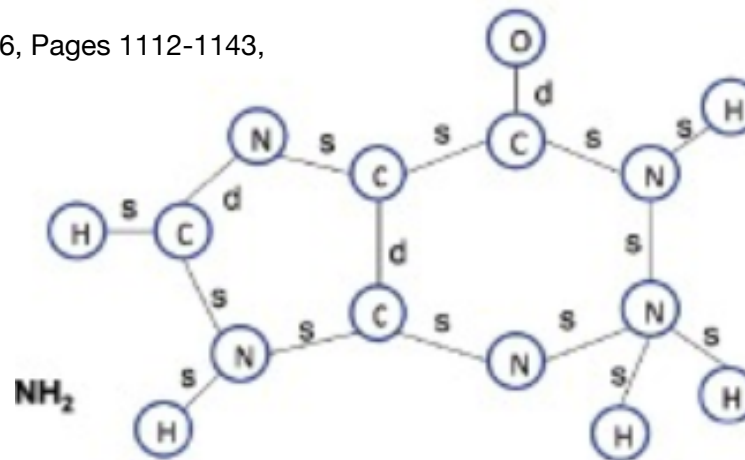
Dipali Pal, Praveen Rao, Vasil Slavov, Anas Katib,

Journal of Computer and System Sciences, Volume 82, Issue 6, 2016, Pages 1112-1143,

<https://doi.org/10.1016/j.jcss.2016.04.002>.

Fig. 1(b), **undirected graphs with edge labels**  
to model chemical compounds such as Guanine and Uracil.  
The vertices of each graph denote the atoms in the  
compound, and the edge labels denote the type of bond  
between two atoms.

example  
for tests





**Fast processing of graph queries on a large database of small and medium-sized data graphs,**

Dipali Pal, Praveen Rao, Vasil Slavov, Anas Katib,

Journal of Computer and System Sciences, Volume 82, Issue 6, 2016, Pages 1112-1143,

<https://doi.org/10.1016/j.jcss.2016.04.002>.

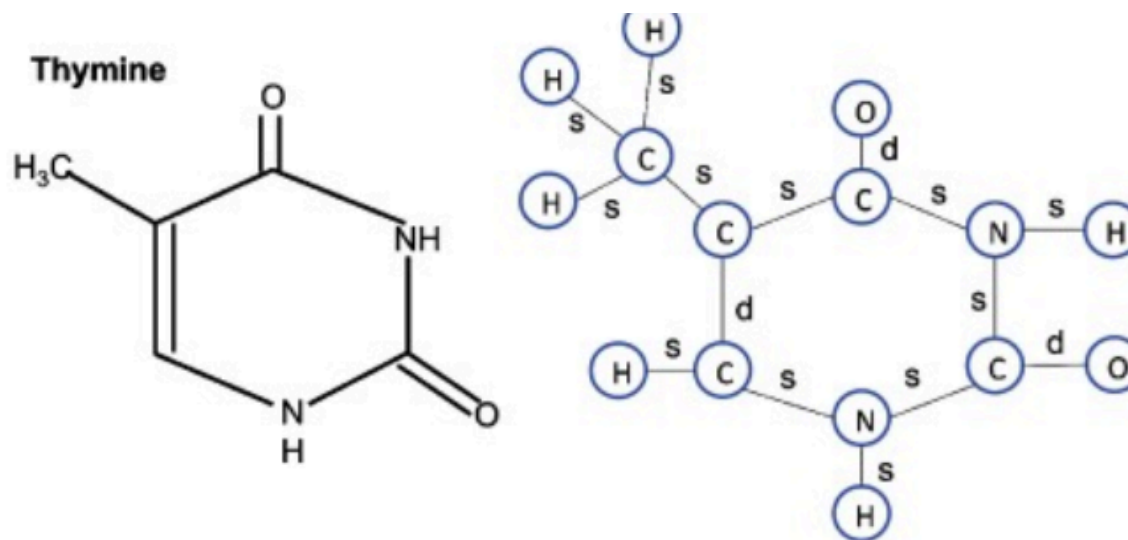
Consider the query in **Fig. 3(b)** for **approximate (full) graph matching on the chemical compounds in Fig. 1(b)**.

Suppose we want to find the compounds that are similar to Thymine within an edit distance of 7 (assuming unit cost for each edit operation). Uracil shown in Fig. 1(b) is a match for the query as one vertex relabel, three vertex additions, and three edge insertions are needed to transform Uracil to Thymine.

Note that Uracil and Thymine belong to a particular family of compounds called Pyrimidine.

example  
for tests

**NOTE: can compare results with Riesen's Graph Matching Toolkit, but the algorithm choices are A-star, A\*-beam search, and A\*-path length. The later can be used on graphs with  $|V| > 12$ .**



(b) Approximate (full) graph matching query  
(chemical compounds)

Fig. 3. Examples of query graphs on real datasets.