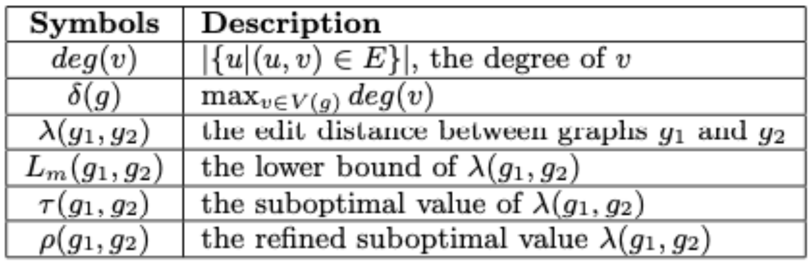
Comparing Stars: On Approximating Graph Edit Distance
Zeng , Tung , Wang , Feng , & Zhou

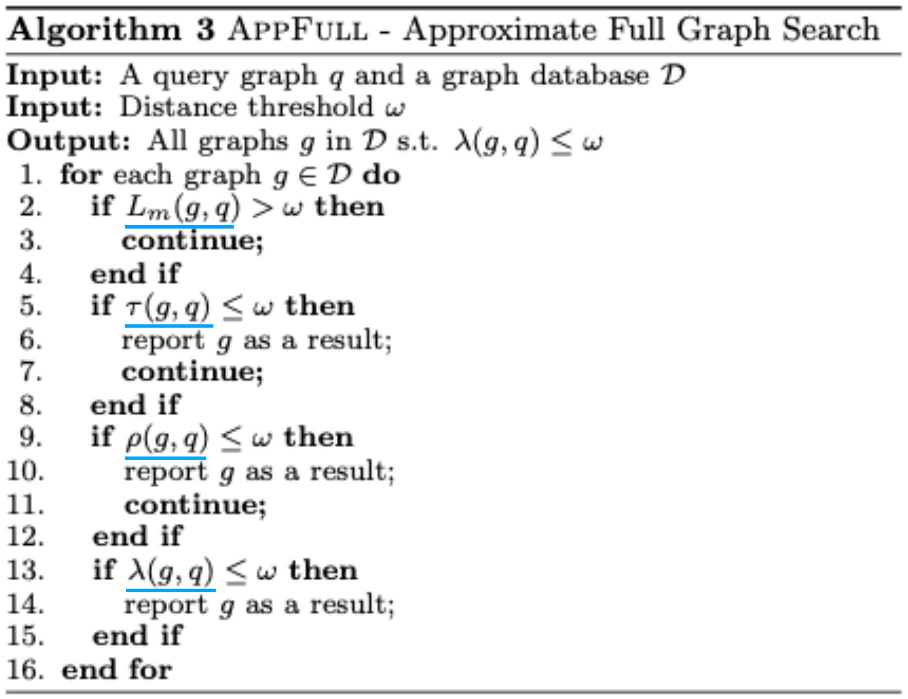


Symbols	Description
$deg(v)$	$\|\{u\|(u,v) \in E\}\|$, the degree of v
$\delta(g)$	$\max_{v \in V(g)} deg(v)$
$\lambda(g_1, g_2)$	the edit distance between graphs g_1 and g_2
$L_m(g_1, g_2)$	the lower bound of $\lambda(g_1, g_2)$
$\tau(g_1, g_2)$	the suboptimal value of $\lambda(g_1, g_2)$
$\rho(g_1,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

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

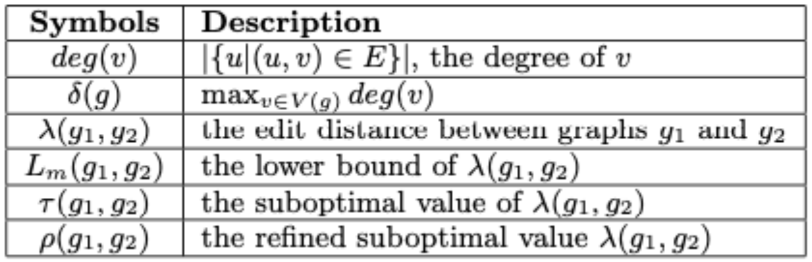
Given a query graph q and the edit distance threshold ω,
for each graph g in graph database D,
L_m(g, q) is first used to filter g if L_m is greater than ω (lines 2-4), because λ > ω must hold in the case of L_m > ω. Subsequently, if τ(g, q) ≤ ω, we know that the edit distance between g and q must be no greater than ω, and q can therefore be reported as a result(lines 5-8). If g passes the above two tests, then ρ is exploited further. Like in the case of τ, if ρ(g,q) ≤ ω, λ(g,q) must be no greater than ω and g can be output as a result (lines 9-12). Finally, if g passes all the above three tests, then λ(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 ρ is the most expensive. Therefore, if g does not pass an earlier test, the rest of the expensive tests are avoided.



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

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 $\lambda(g_1, g_2)$

<u>class diagram for the AppFull and AppSub algorithms</u>

```
ApproxGraphSearchZeng
==========================
————————————————-
createStarStructure(g, i) :  StarStructure

//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, Graph db, double w) : Set<Graph>

//AppSub for performing approx subgraph search.
approxSubGraphSearch() :

// Lemma 4.1
calculateEditDistance(StarStructure s1, StarStructure s2) : double
—————————————
```

```
StarStructure
==============
root : int // index of root
leaves : int[] // the edge vertexes of V[i] treated as leaves
labels:  double[] // the labels as costs
—————————-
————————
```

Comparing Stars: On Approximating Graph Edit Distance
Zeng , Tung , Wang , Feng , & Zhou

Symbols	Description
$deg(v)$	$\lvert\{u \lvert (u,v) \in E\}\rvert$, 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 $\lambda(g_1, g_2)$

needed for AppFull (the approx full graph search

Lower Bound of Graph Edit Distance

Based on Lemma 4.2, μ provides a lower bound L_m of λ, i.e.,

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

algorithm in Section 4.2.1 leads to a mapping $\bar{P}$ from $V(g)$ to $V(h)$, we can simply use Equation 1 from Section 3.1 to compute $C(g, h, \bar{P})$, denoted as $\tau(g, h)$. Apparently, since

mutation matrix P, the cost of transforming g to h using P, $C(g, h, P)$, is defined as

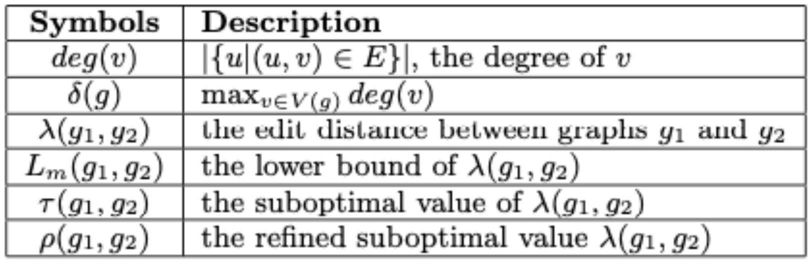
$$C(g, h, P) = \sum_{i=1}^{n} \sum_{j=1}^{n} C_{i,j} P_{i,j} + \frac{1}{2} \lvert\lvert A^g - PA^h P^T \rvert\rvert_1 \qquad (1)$$

$\lambda(g, q)$ **is in Lemma 4.1**

Algorithm 2 REFINE(g,h,P)

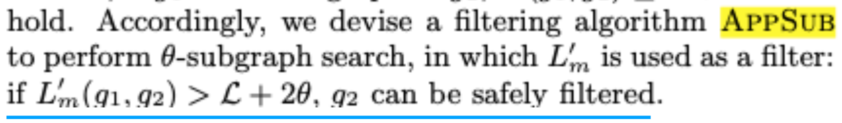
```
Input: two graph structures $g$ and $h$
Input: a permutation matrix $P$ of $g$ and $h$
Output: refined suboptimal distance of $g$ and $h$
 1. $dist \leftarrow C(g, h, P)$;
 2. $min \leftarrow dist$;
 3. for any pair $(u_i, u_j) \in V(g)$ do
 4.     get $P'$ based on $u_i$ and $u_j$;
 5.     if $min > C(g, h, P')$ 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.     $min \leftarrow$ REFINE$(g, h, P_{min})$
12. end if
13. return $min$;
```

Comparing Stars: On Approximating Graph Edit Distance
Zeng , Tung , Wang , Feng , & Zhou



Symbols	Description
$deg(v)$	$\lvert\{u\lvert(u,v)\in E\}\rvert$, 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 $\lambda(g_1,g_2)$

needed for AppSub (the approx sub graph search



hold. Accordingly, we devise a filtering algorithm AppSub to perform θ-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 g1 is subgraph isomorphic to g2, no vertex relabelling will exists in the optimal alignment that make g2 reach g1, 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 g3 is the maximal common subgraph between a query graph g1 and a data graph g2 ,
the number of edge relaxations in Grafil is defined as |E1 |−|E3 |.
*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 g1 is said to be θ-subgraph isomorphic to g2 if there exists a graph g3 s.t.
g3 ⊑ g2 (i.e. ***g3 is a subset of g2***) and ***λ'(g1, g3) ≤ θ***.

Given a query g1 and a graph database D, the problem of θ-subgraph search is to find out all graphs g2 in D of which g1 is a θ-subgraph.

if g1 is a θ-subgraph of g2, λ'(g1, g2) ≤ L + 2θ must hold
AppSub inherently supports both of two kinds of subgraph search, i.e., traditional subgraph search[37] and containment search[9