Figures for SING

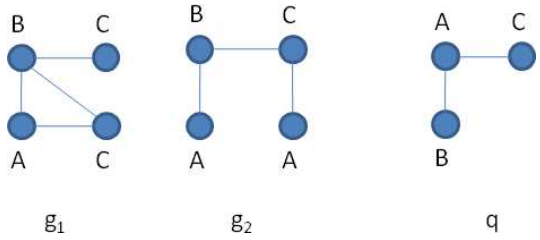

Figure 1: A database of two graphs g_1, g_2 and a query q. $q \precsim g_1$ but $q \not\precsim g_2$.

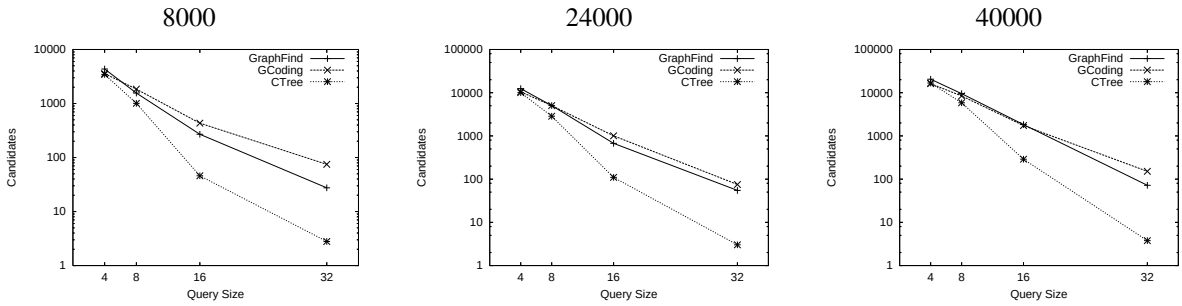

Figure 2: Number of candidates over databases of molecular compounds

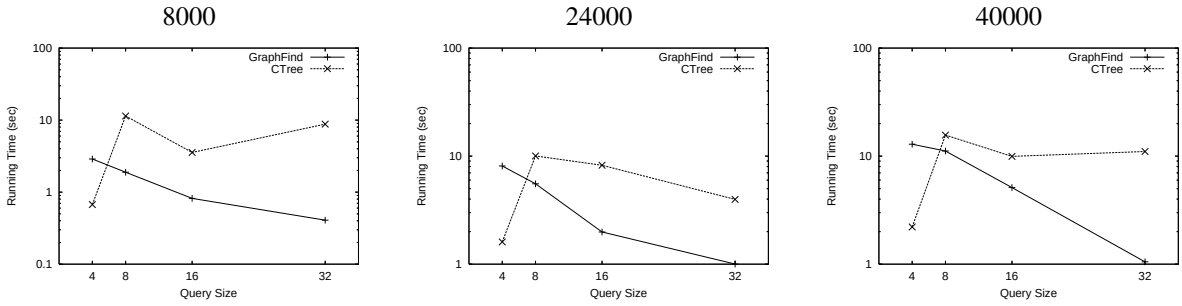

Figure 3: Total query time over databases of molecular compounds. The tools solve the First_query_occurrence problem.

References

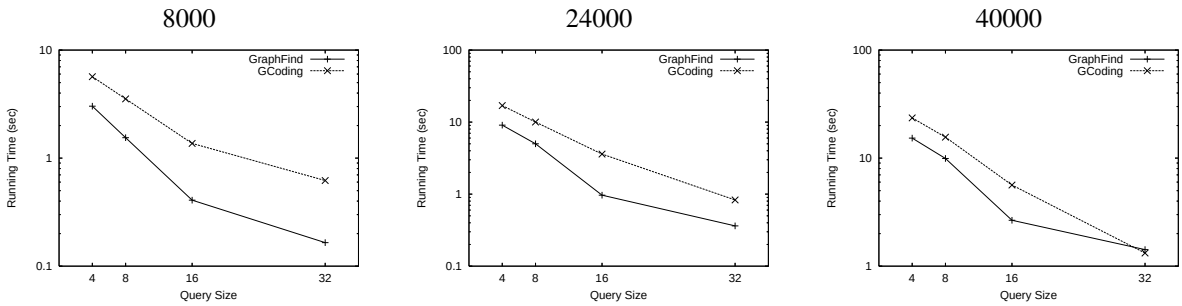

Figure 4: Total query time over databases of molecular compounds. The tools solve the All_query_occurrences problem.

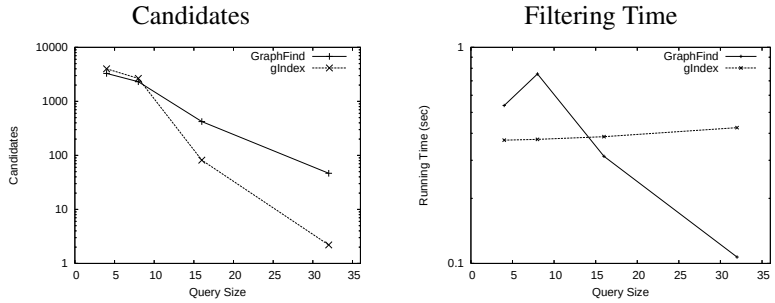

Figure 5: Comparison with gIndex over a dataset of 8000 small molecular compounds.

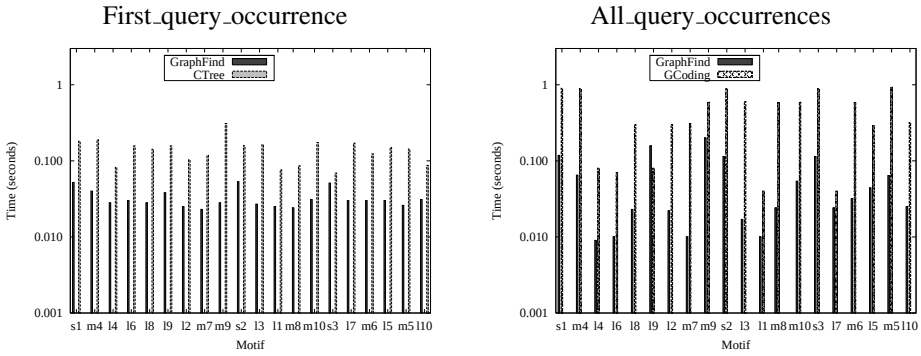

Figure 6: Query time over biological networks

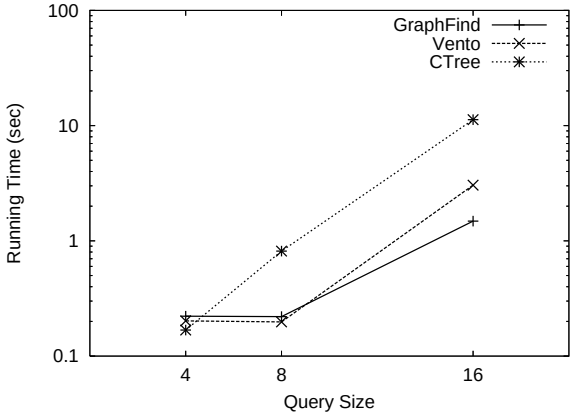

Figure 7: Query time over a single large graph

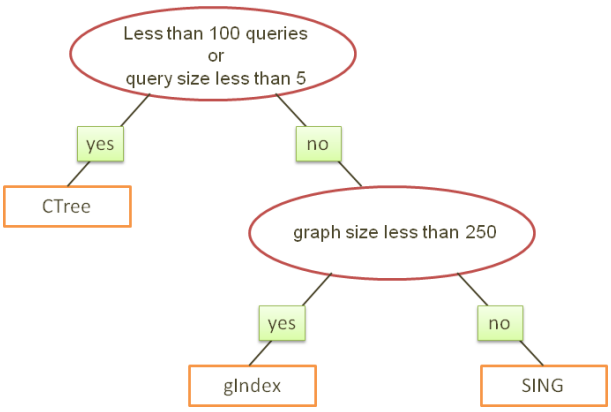

Figure 8: A decision tree showing the best performing system on different scenarios. SING is the best choice on applications where a lot of queries need to be processed on databases of medium and large graphs.

Preprocessing

Input:
database of graphs $D = \{g_1, g_2, ..., g_n\}$
Output:
a global index GI
a local index for each graph $LI[g]$
Algorithm:
$GI \leftarrow \phi$
for each $g = (V, E, \Sigma, l) \in D$ do
$LI[g] \leftarrow \phi$
for each $v \in V$ do
$mark(v) \leftarrow unmarked$
for each $v \in V$ do
$p \leftarrow (l(v))$
$mark(v) \leftarrow marked$
dfs_preprocessing(g,v,p,GI,$LI[g]$)
$mark(v) \leftarrow unmarked$

dfs_preprocessing

Input:
a graph $g = (V, E, \Sigma, l)$
a vertex $v \in V$
a path feature $p = (a_1, a_2, ..., a_{|p|})$
a global index GI
a local index $LI[g]$
Output:
the updated global index GI
the updated local index $LI[g]$
Algorithm:
$GI[p][g] \leftarrow GI[p][g] + 1$
$LI[g][p][v] = 1$
if $|p| = l_p$ than
return
for each $(v, u) \in E | mark(u) = unmarked$ do
$p \leftarrow (a_1, a_2, ..., a_{|p|}, l(u))$
$mark(u) \leftarrow marked$
dfs_preprocessing(g,v,p,GI,$LI[g]$)
$mark(u) \leftarrow unmarked$

Figure 9: Preprocessing

Filtering

Input:
a query graph $q = (V_q, E_q, \Sigma, l_q)$
a database of graphs D
a global index GI
a local index for each graph $LI[g]$
Output:
a set of candidates C_2
a map of vertices compatibility for each candidate graph $M[g]$
Algorithm:
// extract query features
$FQ \leftarrow \phi$
$FVQ \leftarrow \phi$
for each $v \in V_q$ do
$mark(v) \leftarrow unmarked$
for each $v \in V_q$ do
$p \leftarrow (l(v))$
$mark(v) \leftarrow marked$
dfs_filtering(g,v,p,FQ,FVQ)
$mark(v) \leftarrow unmarked$

// first step filtering
$C_1 \leftarrow D$
for each $f \in FQ$ do
$S \leftarrow \phi$
for each $g \in GI[f]$ do
if $GI[f][g] \geq FQ[f]$ then
$S \leftarrow S \cup \{g\}$
$C_1 \leftarrow C_1 \cap S$

// second step filtering
$C_2 \leftarrow C_1$
for each $g \in C_1$ do
for each $v \in V_q$ do
$M[g][v] = \bigcap_{f \in FVQ[v]} LI[g][f]$
if $M[g][v] = \phi$ than
$C_2 \leftarrow C_2 - \{g\}$

dfs_filtering

Input:
a query graph $q = (V_q, E_q, \Sigma, l_q)$
a vertex $v \in V$
a path feature $p = (a_1, a_2, ..., a_{|p|})$
a global query index GIQ
a local query index LIQ
Output:
the updated global query index GIQ
the updated local query index LIQ
Algorithm:
if $|p| = l_p$ OR $\nexists (v, u) \in E | mark(u) = unmarked$ than
$FQ[p] \leftarrow FQ[p] + 1$
$FVQ[v] = FVQ[v] \cup \{p\}$
return
for each $(v, u) \in E | mark(u) = unmarked \in D$ do
$p \leftarrow (a_1, a_2, ..., a_{|p|}, l(u))$
$mark(u) \leftarrow marked$
dfs_filtering(g,v,p,FQ,FVQ)
$mark(u) \leftarrow unmarked$

Figure 10: Preprocessing