TEDI: Efficient Shortest Path Query Answering on Graphs

Fang Wei

University of Freiburg

SIGMOD 2010

Applications

Shortest Path Queries

A shortest path query on a(n) (undirected) graph finds the shortest path for the given source and target vertices in the graph.

1. ranked keyword search
2. XML databases
3. bioinformatics
4. social network
5. ontologies

State-of-the-art Research

Shortest Path

- Concept of *compact BFS-trees* (*Xiao et al. EDBT09*) where the BFS-trees are compressed by exploiting the symmetry property of the graphs.
- Dedicated algorithms specifically on GIS data. It is unknown, whether the algorithms can be extended to dealing the other graph datasets.

State-of-the-art Research

Reachability Query Answering

Well studied in the DB community

- *2-HOP approach*: pre-compute the transitive closure, so that the reachability queries can be more efficiently answered comparing to BFS or DFS.
- *interval labeling approach*: first extract some tree from the graph, then store the transitive closure of the rest of the vertices.

State-of-the-art Research

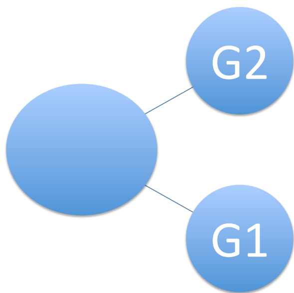
Reachability Query Answering

Well studied in the DB community

- *2-HOP approach*: pre-compute the transitive closure, so that the reachability queries can be more efficiently answered comparing to BFS or DFS.
- *interval labeling approach*: first extract some tree from the graph, then store the transitive closure of the rest of the vertices.

Can not be extended to cope with the shortest path query answering: require only a boolean answer (yes or no); the transitive closure stored in the index can be drastically compressed.

TEDI: Intuition of decomposing graphs



- Subgraphs G_1 and G_2 are connected through a small set of vertices S.
- Then any shortest path from $u \in G_1$ to $v \in G_2$ has to pass through some vertex $s \in S$.
- Do it recursively in G_1 and G_2.

TEDI: our approach

TEDI (TreE Decomposition based Indexing)

- an indexing and query processing scheme for the shortest path query answering.
- we first *decompose* the graph G into a tree in which each node contains a set of vertices in G.
- there are overlapping among the bags
- connectedness of the tree

TEDI: our approach

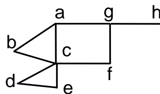
TEDI (TreE Decomposition based Indexing)

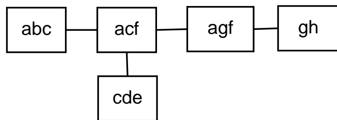
- Based on the tree index, we can execute the shortest path search in a *bottom-up* manner and the query time is decided by the height and the bag cardinality of the tree, instead of the size of the graph.
- pre-compute the *local* shortest paths among the vertices in every bag of the tree.

Tree Decomposition

Tree Decomposition

1. Tree with a vertex set (bag) associated with every node
2. For every edge (v, w): there is a bag containing both v and w
3. For every v: the bags that contain v form a connected subtree

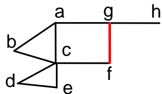


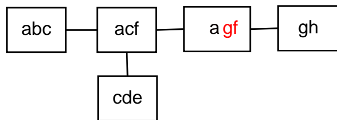


Tree Decomposition

Tree Decomposition

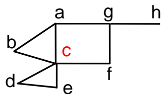
1. Tree with a vertex set (bag) associated with every node
2. For every edge (v, w): there is a bag containing both v and w
3. For every v: the bags that contain v form a connected subtree

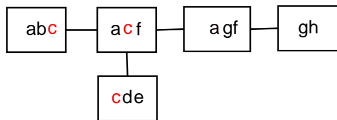




Tree Decomposition

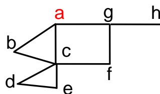
1. Tree with a vertex set (bag) associated with every node
2. For every edge (v, w): there is a bag containing both v and w
3. For every v: the bags that contain v form a connected subtree

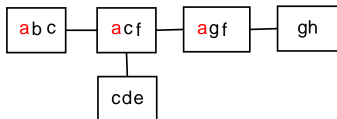




Tree Decomposition

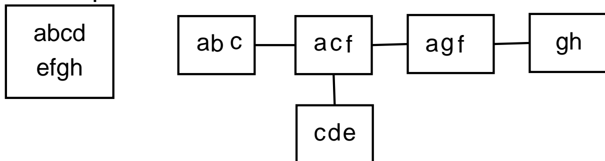
1. Tree with a vertex set (bag) associated with every node
2. For every edge (v, w): there is a bag containing both v and w
3. For every v: the bags that contain v form a connected subtree



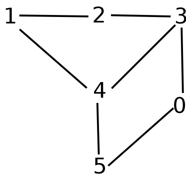


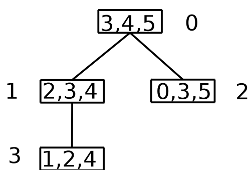
Treewidth

- The width of a tree decomposition T_G is its maximal bag size (cardinality).
- The treewidth of G is the minimum width over all tree decompositions of G.



Example of tree decomposition



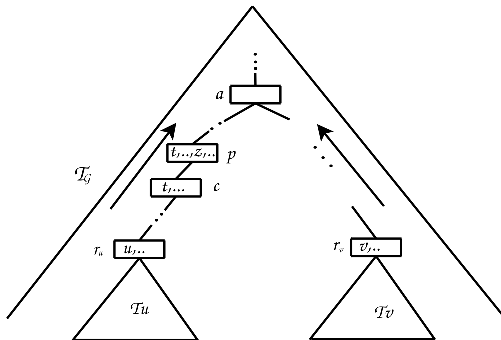


- Treenode: a pair (n, b) where $n \in G$ and b is the bag number in T_G.
- There is a path from u to v in G iff there is a treepath from $(u, *)$ to $(v, *)$.
- Treepath is composed of Inner edges (eg. $((1, 3), (2, 3))$) and Inter edges (e.g. $((2, 3), (2, 1))$).

Shortest path over TD

- The Intuition: restricting the search space of the vertices in the shortest path from u to v.
- For every vertex u in G, there is an *induced subtree* of u: r_u.
- Idea: checking the shortest distance from u (v) to the vertices in the bags along the *simple path* from r_u to r_v.

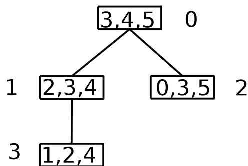
Shortest path over TD



Correctness intuition: every path from u to v passes through all the bags in the simple path from r_u to r_v.

Shortest path over TD

- Compute the shortest distances from r_u (r_v) to the youngest common ancestor in a bottom-up manner.
- Pre-computation of the local shortest distances in every bag.



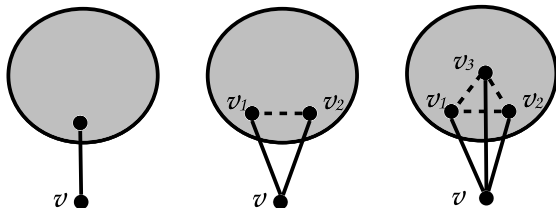
Shortest path over TD: Complexity

- Query: $O(tw^2 h)$, tw is the bag candinality, and h the height of the tree decomposition.
- Index construction:
  1. Decomposing graph: $O(n)$ (see heuristic algorithm later)
  2. Local shortest paths computation $O(n^2)$

Tree Decomposition Algorithm

- NP-complete for the problem of given constant k, whether there exists a tree decomposition for which the treewidth is less then k.
- Heuristics and approximation

Tree Decomposition Algorithm



Definition (Simplicial)

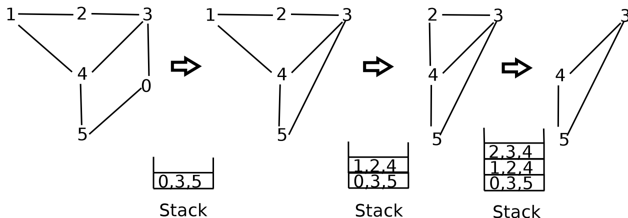
A vertex v is simplicial in a graph G if the neighbors of v form a clique in G.

Theorem

If v is a simplicial vertex in a graph G, then T_G can be obtained from T_{G-v} by increasing the treewidth of maximal 1.

Tree Decomposition Algorithm

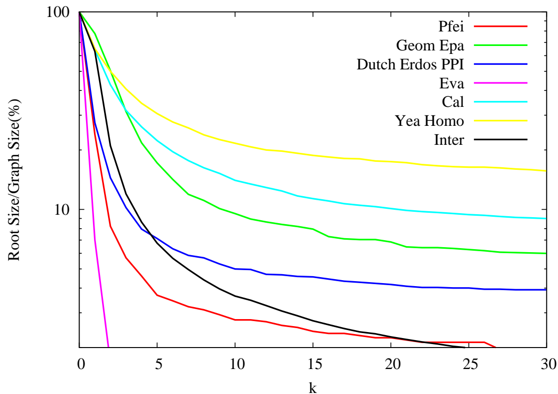
- Each time a vertex v with a specific degree k is identified. First check whether all its neighbors form a clique, if not, add the missing edges to construct a clique.
- Then v together with its neighbors are pushed into the stack, then delete v and the corresponding edges in the graph.
- Continue till either the graph is reduced to an empty set of the upper bound of k is reached.



Algorithm Improvement

- Problem of the tree decomposition with big root size:
  $\rightarrow O(tw^2 h)$ not satisfying.
- Observation: *only* root has big size $|R|$, and the rest bags have the size upper bound of k, which can be tuned in the algroríthm
  $\rightarrow k \ll |R|$
- Query answering algorithm modified: $O(k^2 h)$ instead of $O(tw^2 h)$.
- Trade-off of k and $|R|$.

$k - |R|$ Curve



Experiment (1) Real Data

Graph	n	#TreeN	#SumV	h	k	$\|R\|$
Pfei	1738	1680	3916	16	6	60
Gemo	3621	3000	9985	10	5	623
Epa	4253	3637	11137	7	7	618
Dutsch	3621	3442	8700	9	5	258
Eva	4475	4457	9303	9	2	75
Cal	5925	5095	18591	14	10	832
Erdos	6927	6690	18979	9	7	405
PPI	1458	1359	3638	11	7	101
Yeast	2284	1770	6708	6	9	516
Homo	7020	5778	24359	10	15	1244
Inter	22442	21757	67519	10	13	687

Table: Statistics of real graphs and the properties of the index

Experiment (1) Real Data

	Index Size (MB)				Index Time (s)			
Graph	paths	tree	TEDI	SYMM	t_{tree}	t_{paths}	TEDI	SYMM
Pfei	0.025	0.008	0.033	7.9243	0.003	0.099	0.102	2.688
Gemo	1.81	0.020	1.830	44.9907	0.068	0.878	0.946	14.859
Epa	1.63	0.022	1.652	28.1992	0.056	0.97	1.026	37.14
Dutsch	0.404	0.016	0.420	20.8559	0.011	0.311	0.322	13.687
Eva	0.026	0.018	0.044	5.5447	0.006	0.239	0.245	289.532
Cal	3.04	0.038	3.078	92.026	0.145	2.535	2.680	34.094
Erdos	0.516	0.018	0.534	32.2695	0.038	0.849	0.887	90.453
PPI	0.052	0.008	0.060	5.954	0.004	0.130	0.134	1.547
Yeast	1.08	0.014	1.094	19.4457	0.019	0.566	0.585	7.578
Homo	6.88	0.048	6.928	21.574	0.198	7.745	7.943	53.985
Inter	1.66	0.136	1.796	744.07478	0.796	15.858	16.654	1709.64

Table: Comparison between TEDI and SYMM on index construction of real dataset.

Experiment (1) Real Data

	TEDI			SYMM
Graph	TEDI (ms)	BFS	Speedup	Speedup
Pfei	0.003420	0.052	15.2	13.04
Gemo	0.002933	0.123	42.4	41.10
Epa	0.002096	0.105	50.0	39.62
Dutsch	0.002655	0.097	37.3	28.21
Eva	0.002299	0.089	38.7	20.20
Cal	0.003325	0.187	56.7	59.31
Erdos	0.002037	0.146	71.9	57.72
PPI	0.002629	0.050	19.2	13.30
Yeast	0.002463	0.071	28.4	25.63
Homo	0.007666	0.226	29.7	N.a.
Inter	0.004178	0.693	169.0	N.a.

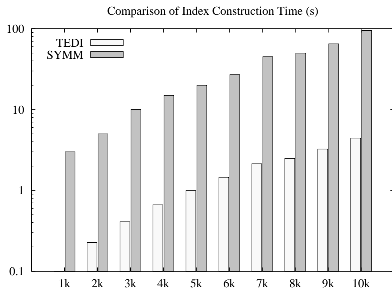
Table: Comparison between TEDI and SYMM on query time over real dataset.

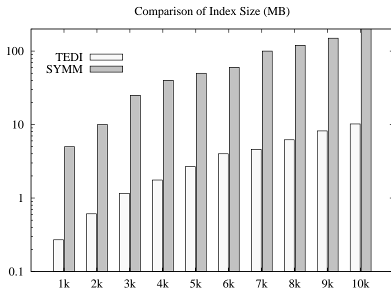
Experiment (2) Synthetic Data

Graph	n	#TreeN	#SumV	h	k	$\|R\|$
1k	1000	808	2131	9	3	194
2k	2000	1730	4786	11	5	272
3k	3000	2641	7362	10	6	361
4k	4000	3559	10131	10	7	443
5k	5000	4460	12758	10	8	542
6k	6000	5355	15371	10	9	612
7k	7000	6292	18626	12	9	710
8k	8000	7201	20790	11	9	801
9k	9000	8089	23497	12	9	913
10k	10000	8983	26224	11	9	1019

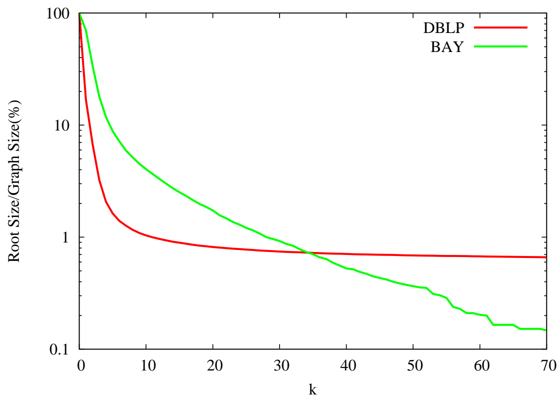
Table: Statistics of the synthetic graphs and the properties of the index

Experiment (2) Synthetic Data





Experiment (3) Scalability over Large Datasets



Experiment (3) Scalability over Large Datasets

Graph	n	#TreeN	#SumV	h	k	$\|R\|$
DBLP	592 983	589 164	1 309 710	30	100	3821
BAY	321 272	321 028	1 298 993	351	80	245

Table: Statistics of large graphs and the properties of the index

Experiment (3) Scalability over Large Datasets

	Index Size (MB)			Index Time (s)		
Graph	paths	tree	TEDI	t_{tree}	t_{paths}	TEDI
DBLP	117.2	2.6	119.8	102.4	2124.0	2226.4
BAY	24.7	2.6	27.3	182.2	2859.7	3041.9

Table: Index construction of large dataset.

	Query Time		
Graph	TEDI (ms)	BFS (ms)	Speedup
DBLP	0.055	32.47	590.0
BAY	0.258	20.54	80.0

Table: Comparison of TEDI query time on large datasets to BFS

Conclusion

Main Results

- An index structure based on tree decomposition for answering shortest path queries over (un)directed graph.
- Efficiency on query answering, index construction.
- Can be extended to weighted graphs: query answering remains same, takes longer time for index construction.

Conclusion

Main Results

- An index structure based on tree decomposition for answering shortest path queries over (un)directed graph.
- Efficiency on query answering, index construction.
- Can be extended to weighted graphs: query answering remains same, takes longer time for index construction.

Future Work

- Ranked keyword search over graph data.