Treewidth Engineering: Shortest Path and kNN Query Answering using Tree Decomposition Techniques

Fang Wei

University of Freiburg, Germany

October 18, 2010

Outline

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. bioinformatics
3. social network
4. ontologies

Why the classic algorithms are not good enough?

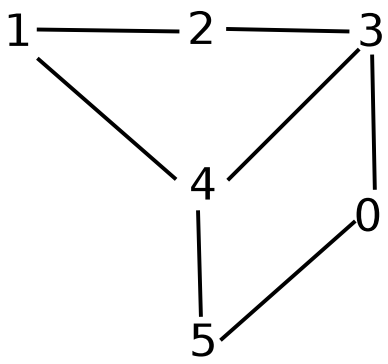
1. scalibility over large datasets (facebook, genenetworks)
2. accumulation of many queries

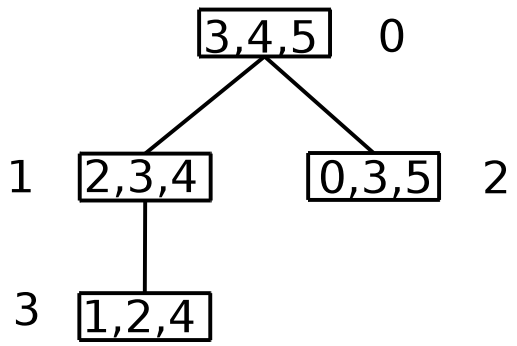
TEDI: our approach

TEDI (TreE Decomposition based Indexing)

- an indexing and query processing scheme for the shortest path query answering.
- we first decompose graph G.
- Based on the structure, we can execute the shortest path search in a *bottom-up* manner and the query time is decided by the height and the treewidth of the graph, instead of the size of the graph.
- pre-compute the *local* shortest paths among the vertices in every bag of the tree.

Example of tree decomposition



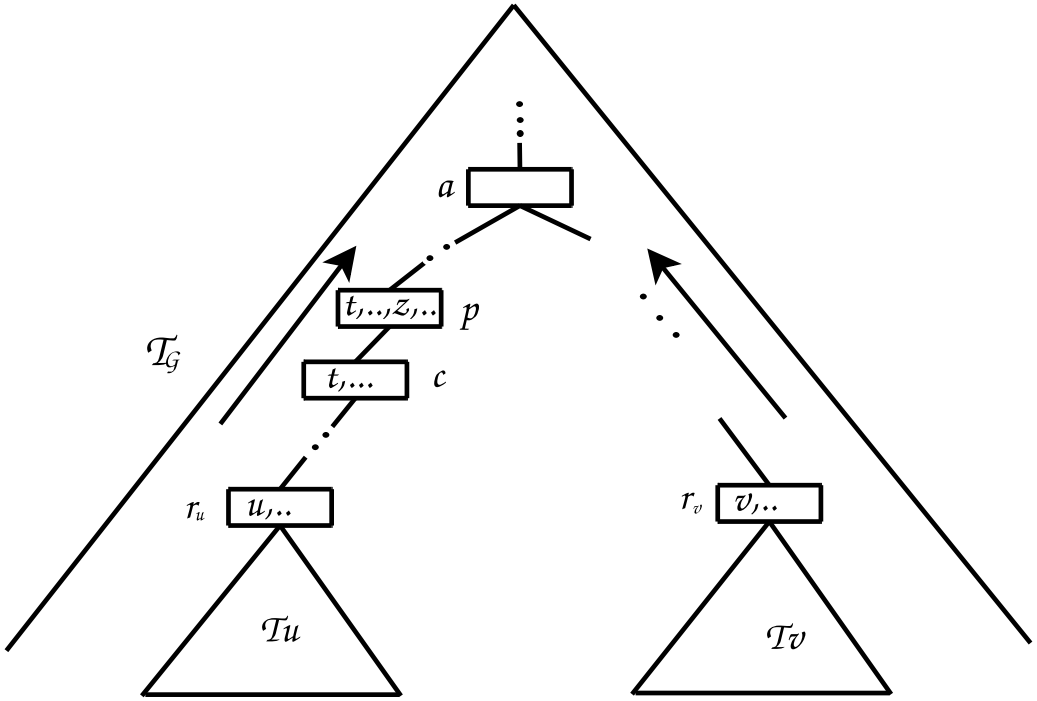


- Treenode: a pair (n, b) where $n \in G$ and b is the bag number in T_G.
- There is a path from u to v in G if and only if there is a treepath from $(u, *)$ to $(v, *)$.
- Treepath is composed of Inner edges (eg. $((1,3),(2,3))$) and Inter edges (e.g. $((2,3),(2,1))$).

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.

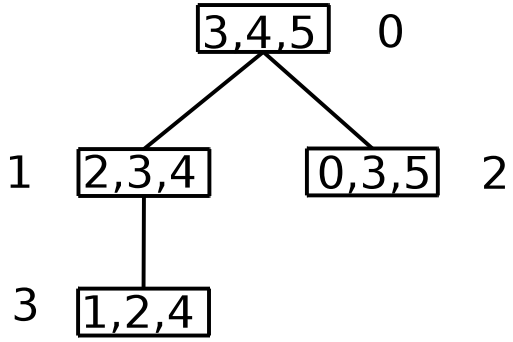
Shortest path over TD



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.



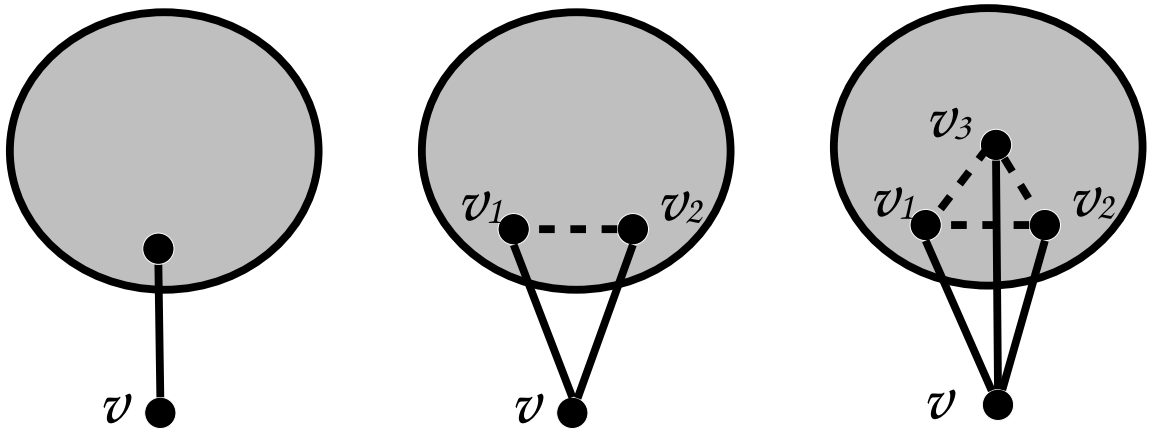
Shortest path over TD: Complexity

- 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)

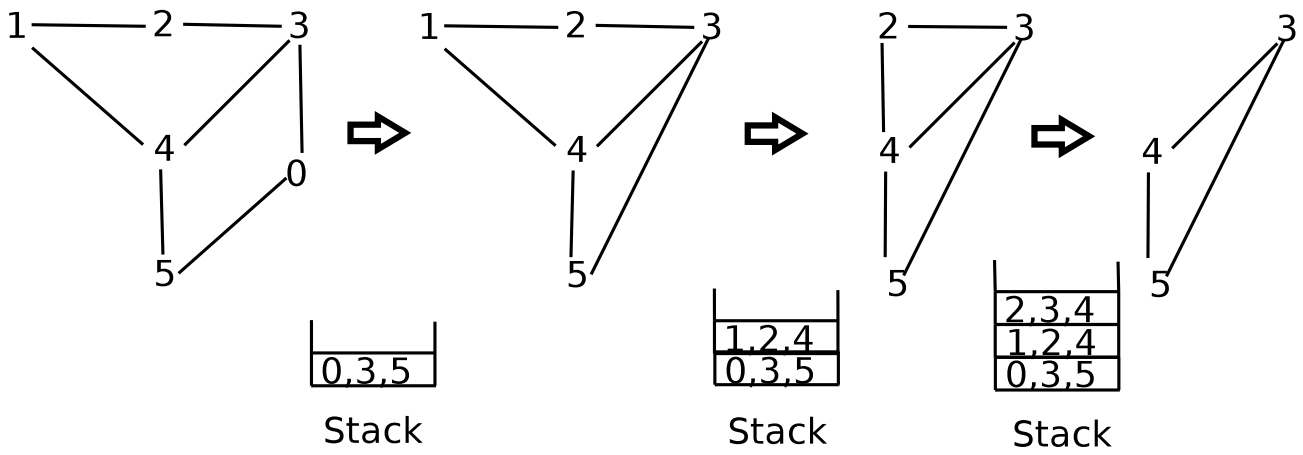
A vertex v is simplicial in a graph G if the neighbors of v form a clique in G.

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

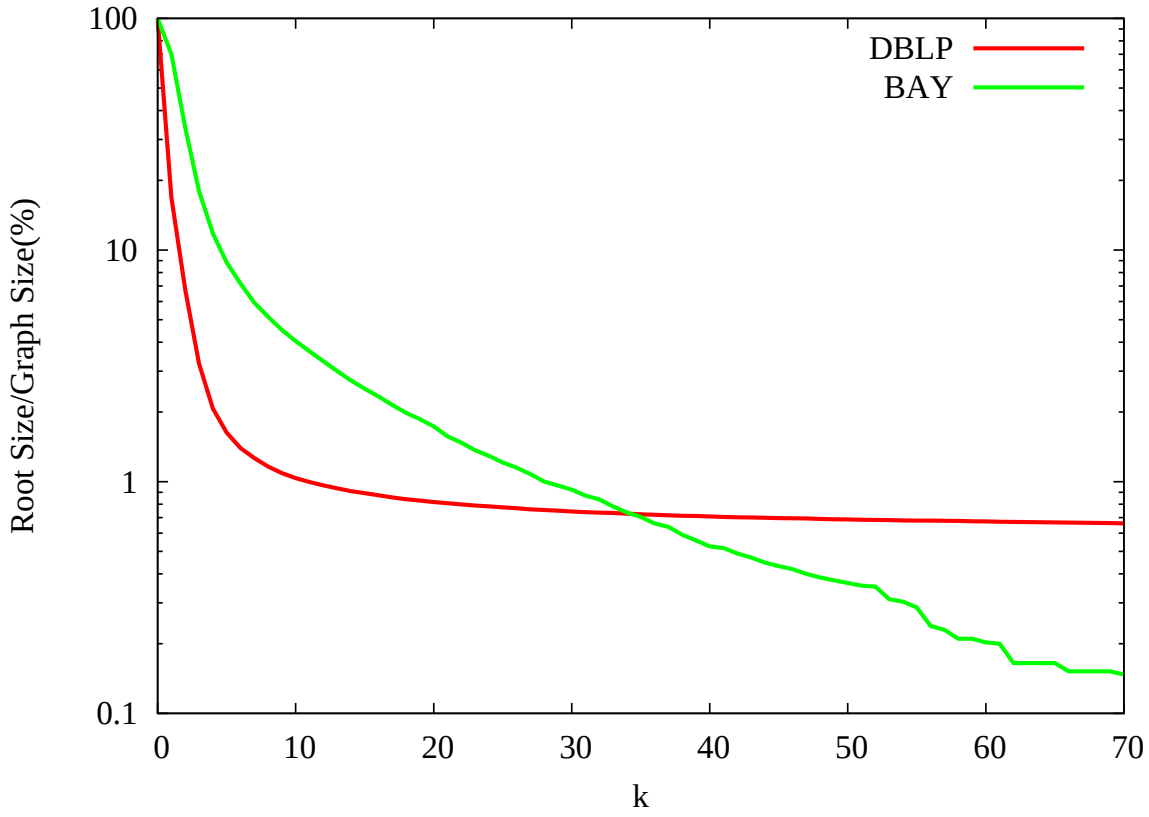
- Each time a vertex v with a specific degree k is identified. First check whether all its neighbors form a clique, if not, add the missing edges to construct a clique.
- Then v together with its neighbors are pushed into the stack, then delete v and the corresponding edges in the graph.
- Continue till 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 algrorithm
  $\rightarrow k \ll |R|$
- Query answering algorithm modified: $O(k^2 h)$ instead of $O(tw^2 h)$.
- Trade-off of k and $|R|$.

$k - |R|$ Curve



Experiment over Large Datasets

Graph	n	#TreeN	#SumV	h	k	$\lvert R \rvert$
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 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

k Nearest Neighbor Query Answering

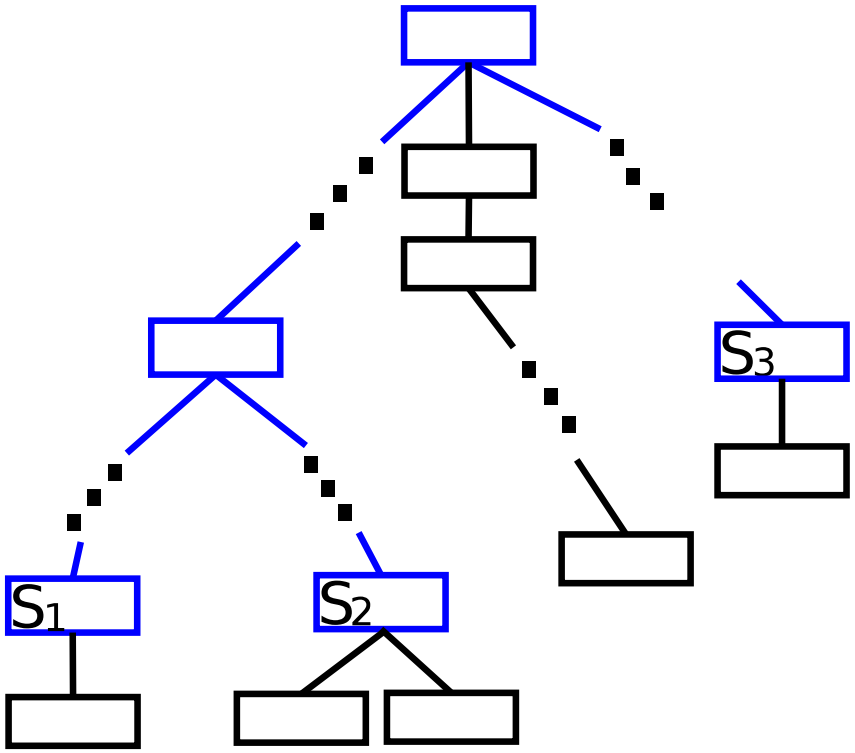
In a road network, a user usually wants to find k nearest Petrol stations (restaurants etc.) from her/his current location.

Definition (k Nearest Neighbor Query)

Let $G = (V, E)$ be a weighted graph with $|V| = N$ and $|E| = M$. Let $u \in V$ and $S \subseteq V$. The k Nearest Neighbor (kNN) query of u with respect to S returns k nearest neighbor vertices S' of u, such that $S' \subseteq S$.

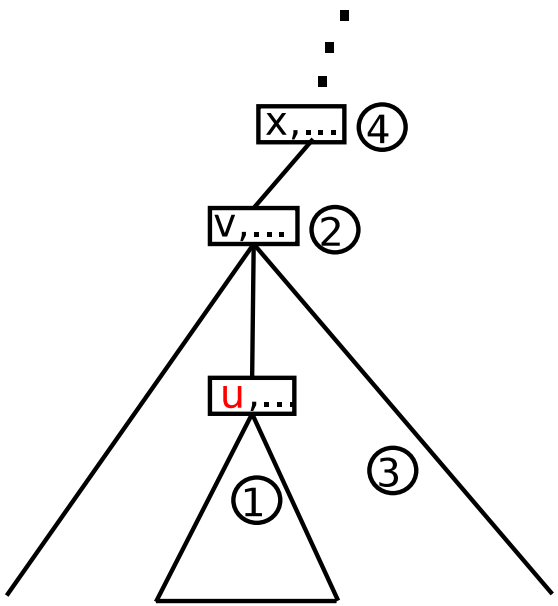
Time cost of Dijkstra's Algorithm:
$k/|S| \cdot M \log N$

kNN Query over TD



Step 1: S-pruning. Prune the subtrees which do not contain any *S* vertex.

kNN Query over TD



Step 2: Traversing from the induced subtree root of u, top-down $\rightarrow$ one step up $\rightarrow$ top-down,...$\rightarrow$root $\rightarrow$top-down.

- maintain a list of current k nearest neighbors, where the kth neighbor is the upper bound ub.
- compute the shortest paths from u to the vertices v in the bags.
- if $sdist(u, v) > ub$, v is removed from the current bag.
- stop condition: current bag is empty.

Experiment over Large Datasets

Graph	n	#TreeN	#SumV	h	k	$\lvert R \rvert$
USA-NW	1 207 945	1 207 383	4 285 201	424	80	562

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

	Query Time		
Graph	k	TEDI-kNN (ms)	Dijkstra (ms)
USA-NW	10	1.5	13.6
USA-NW	30	3.3	39.1
USA-NW	50	4.3	69.0
USA-NW	70	5.0	99.5
USA-NW	90	5.9	131.6

Table: Comparison of kNN query time with $|S| = 6000$ to Dijstra Algorithm

Lessons Learnt

- Tree decomposition-bases techniques can be applied to solving tractable problems more efficiently.
- Bounded treewidth is not a must.
- More investigation on practical tree decomposition algorithms over large scale graphs (scalibility rules).