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

Fang Wei

University of Freiburg, Germany

October 18, 2010
Outline

1. Motivation
2. Shortest Path Query Answering
3. Tree Decomposition Algorithm
4. k Nearest Neighbor Query Answering
5. Conclusion
Applications

Shortest Path Queries
A shortest path query on an (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. scalability 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** (e.g. ((1, 3), (2, 3))) and **Inter edges** (e.g. ((2, 3), (2, 1))).
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 $r_v$. 
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: $O(tw^2h)$, $tw$ is the bag cardinality, 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 than $k$.
- Heuristics and approximation
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 or the upper bound of $k$ is reached.
Algorithm Improvement

- Problem of the tree decomposition with big root size:
  \[ O(tw^2h) \] 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 algorithm
  \[ k \ll |R| \]
- Query answering algorithm modified: \(O(k^2h)\) instead of \(O(tw^2h)\).
- Trade-off of \(k\) and \(|R|\).
$k - |R|$ Curve

![Graph showing the relationship between $k$ and $|R|$ with two curves, one for DBLP and one for BAY. The graph plots Root Size/Graph Size (%) on the y-axis against $k$ on the x-axis. The DBLP curve is red and the BAY curve is green.]
### Experiment 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 over Large Datasets

<table>
<thead>
<tr>
<th></th>
<th>Index Size (MB)</th>
<th>Index Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>paths</td>
<td>tree</td>
</tr>
<tr>
<td>DBLP</td>
<td>117.2</td>
<td>2.6</td>
</tr>
<tr>
<td>BAY</td>
<td>24.7</td>
<td>2.6</td>
</tr>
</tbody>
</table>

Table: Index construction of large dataset.

<table>
<thead>
<tr>
<th></th>
<th>Query Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TEDI (ms)</td>
</tr>
<tr>
<td>DBLP</td>
<td>0.055</td>
</tr>
<tr>
<td>BAY</td>
<td>0.258</td>
</tr>
</tbody>
</table>

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:

$k/|S| \cdot M \log N$
Step 1: S-pruning. Prune the subtrees which do not contain any S vertex.
Step 2: Traversing from the induced subtree root of \( u \), top-down
→ one step up → top-down,...→root →top-down.
- maintain a list of current \( k \) nearest neighbors, where the
  \( k \)th 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   | $h$  | $k$  | $|R|$ |
|----------|-------|---------|---------|------|------|-------|
| 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

<table>
<thead>
<tr>
<th>Graph</th>
<th>$k$</th>
<th>TEDI-kNN (ms)</th>
<th>Dijkstra (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>USA-NW</td>
<td>10</td>
<td>1.5</td>
<td>13.6</td>
</tr>
<tr>
<td>USA-NW</td>
<td>30</td>
<td>3.3</td>
<td>39.1</td>
</tr>
<tr>
<td>USA-NW</td>
<td>50</td>
<td>4.3</td>
<td>69.0</td>
</tr>
<tr>
<td>USA-NW</td>
<td>70</td>
<td>5.0</td>
<td>99.5</td>
</tr>
<tr>
<td>USA-NW</td>
<td>90</td>
<td>5.9</td>
<td>131.6</td>
</tr>
</tbody>
</table>

**Table:** Comparison of kNN query time with $|S| = 6000$ to Dijkstra 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).