SPDO: High-Throughput Road Distance Computations on Spark Using Distance Oracles

Shangfu Peng
shangfu@cs.umd.edu
University of Maryland
Spatial Tek LLC

Jagan Sankaranarayanan
jagan@nec-labs.com
NEC Labs America

Hanan Samet
hjs@cs.umd.edu
University of Maryland
Spatial Tek LLC

ICDE 2016
Motivation: Need to compute millions of network distances or trip times per second on a road network.
Who needs such computations?

Large Package Delivery Companies

Small Food Delivery Companies

e-commerce companies

Trucking Companies

SCHNEIDER
Delivery companies compute Origin-Destination (OD) matrices that can quickly require million distance computations.

### Distances Between the Maggie's Centres in Miles

<table>
<thead>
<tr>
<th>Maggie's Centres</th>
<th>Highlands</th>
<th>Aberdeen</th>
<th>Dundee</th>
<th>Fife</th>
<th>Forth Valley</th>
<th>Edinburgh</th>
<th>Lanarkshire</th>
<th>Glasgow</th>
<th>Newcastle at the Christie</th>
<th>Merseyside</th>
<th>Nottingham</th>
<th>Chesterfield</th>
<th>Wallace</th>
<th>Oxford</th>
<th>West London</th>
<th>Swansea</th>
</tr>
</thead>
<tbody>
<tr>
<td>Highlands</td>
<td>0</td>
<td>129</td>
<td>132</td>
<td>148</td>
<td>150</td>
<td>161</td>
<td>169</td>
<td>180</td>
<td>277</td>
<td>574</td>
<td>386</td>
<td>439</td>
<td>489</td>
<td>611</td>
<td>619</td>
<td>686</td>
</tr>
<tr>
<td>Aberdeen</td>
<td>129</td>
<td>0</td>
<td>70</td>
<td>96</td>
<td>124</td>
<td>124</td>
<td>141</td>
<td>148</td>
<td>260</td>
<td>262</td>
<td>263</td>
<td>410</td>
<td>488</td>
<td>497</td>
<td>544</td>
<td>563</td>
</tr>
<tr>
<td>Dundee</td>
<td>132</td>
<td>70</td>
<td>0</td>
<td>33</td>
<td>69</td>
<td>57</td>
<td>73</td>
<td>80</td>
<td>182</td>
<td>284</td>
<td>296</td>
<td>343</td>
<td>400</td>
<td>422</td>
<td>430</td>
<td>476</td>
</tr>
<tr>
<td>Fife</td>
<td>148</td>
<td>96</td>
<td>33</td>
<td>0</td>
<td>34</td>
<td>26</td>
<td>50</td>
<td>67</td>
<td>161</td>
<td>359</td>
<td>270</td>
<td>317</td>
<td>374</td>
<td>388</td>
<td>404</td>
<td>450</td>
</tr>
<tr>
<td>Forth Valley</td>
<td>150</td>
<td>124</td>
<td>66</td>
<td>34</td>
<td>0</td>
<td>30</td>
<td>17</td>
<td>24</td>
<td>138</td>
<td>226</td>
<td>240</td>
<td>287</td>
<td>344</td>
<td>368</td>
<td>374</td>
<td>408</td>
</tr>
<tr>
<td>Edinburgh</td>
<td>161</td>
<td>194</td>
<td>97</td>
<td>90</td>
<td>30</td>
<td>0</td>
<td>38</td>
<td>48</td>
<td>190</td>
<td>227</td>
<td>238</td>
<td>298</td>
<td>343</td>
<td>367</td>
<td>372</td>
<td>419</td>
</tr>
<tr>
<td>Lanarkshire</td>
<td>162</td>
<td>141</td>
<td>73</td>
<td>60</td>
<td>17</td>
<td>36</td>
<td>0</td>
<td>14</td>
<td>148</td>
<td>212</td>
<td>224</td>
<td>271</td>
<td>328</td>
<td>365</td>
<td>404</td>
<td>424</td>
</tr>
<tr>
<td>Glasgow</td>
<td>160</td>
<td>148</td>
<td>80</td>
<td>67</td>
<td>24</td>
<td>48</td>
<td>14</td>
<td>0</td>
<td>165</td>
<td>221</td>
<td>232</td>
<td>290</td>
<td>337</td>
<td>368</td>
<td>388</td>
<td>412</td>
</tr>
<tr>
<td>Newcastle</td>
<td>277</td>
<td>250</td>
<td>182</td>
<td>161</td>
<td>136</td>
<td>120</td>
<td>145</td>
<td>166</td>
<td>0</td>
<td>158</td>
<td>161</td>
<td>266</td>
<td>240</td>
<td>270</td>
<td>283</td>
<td>361</td>
</tr>
<tr>
<td>at the Christie</td>
<td>374</td>
<td>352</td>
<td>284</td>
<td>236</td>
<td>226</td>
<td>227</td>
<td>212</td>
<td>221</td>
<td>169</td>
<td>43</td>
<td>43</td>
<td>89</td>
<td>128</td>
<td>181</td>
<td>153</td>
<td>204</td>
</tr>
<tr>
<td>Merseyside</td>
<td>366</td>
<td>383</td>
<td>296</td>
<td>270</td>
<td>240</td>
<td>238</td>
<td>224</td>
<td>232</td>
<td>165</td>
<td>43</td>
<td>0</td>
<td>117</td>
<td>148</td>
<td>189</td>
<td>178</td>
<td>222</td>
</tr>
<tr>
<td>Nottingham</td>
<td>432</td>
<td>410</td>
<td>343</td>
<td>317</td>
<td>287</td>
<td>278</td>
<td>271</td>
<td>279</td>
<td>161</td>
<td>59</td>
<td>117</td>
<td>0</td>
<td>108</td>
<td>96</td>
<td>117</td>
<td>132</td>
</tr>
<tr>
<td>Chesterfield</td>
<td>489</td>
<td>468</td>
<td>400</td>
<td>374</td>
<td>344</td>
<td>343</td>
<td>326</td>
<td>337</td>
<td>296</td>
<td>128</td>
<td>148</td>
<td>108</td>
<td>0</td>
<td>191</td>
<td>42</td>
<td>104</td>
</tr>
<tr>
<td>Wallace</td>
<td>511</td>
<td>489</td>
<td>422</td>
<td>389</td>
<td>288</td>
<td>367</td>
<td>360</td>
<td>368</td>
<td>240</td>
<td>181</td>
<td>199</td>
<td>86</td>
<td>121</td>
<td>0</td>
<td>95</td>
<td>253</td>
</tr>
<tr>
<td>Oxford</td>
<td>519</td>
<td>497</td>
<td>430</td>
<td>404</td>
<td>274</td>
<td>372</td>
<td>359</td>
<td>396</td>
<td>270</td>
<td>168</td>
<td>178</td>
<td>117</td>
<td>42</td>
<td>98</td>
<td>0</td>
<td>53</td>
</tr>
<tr>
<td>West London</td>
<td>586</td>
<td>644</td>
<td>476</td>
<td>460</td>
<td>408</td>
<td>419</td>
<td>404</td>
<td>412</td>
<td>283</td>
<td>204</td>
<td>222</td>
<td>132</td>
<td>104</td>
<td>80</td>
<td>63</td>
<td>139</td>
</tr>
<tr>
<td>Swansea</td>
<td>586</td>
<td>665</td>
<td>496</td>
<td>470</td>
<td>440</td>
<td>439</td>
<td>424</td>
<td>432</td>
<td>361</td>
<td>224</td>
<td>182</td>
<td>203</td>
<td>108</td>
<td>238</td>
<td>159</td>
<td>199</td>
</tr>
</tbody>
</table>

*Please note these distances are based on the most direct driving route and distances will vary depending on mode of travel and route chosen.*

**If you're feeling adventurous you could also visit Maggie's in Hong Kong!**
Uber has 35,000 taxis in NYC and need to process customer requests, analyze GPS track etc.
Local Advertisement companies need to serve ads to people that are both proximal and can reach the customer business, need to do that at say 10K impressions a second!
Large-scale analysis such as counting how many jobs are accessible within 40 minutes of each census block.

State Smart Transportation Initiative (SSTI*) Example

*source: http://www.ssti.us/Events/accessibility-towards-a-new-multimodal-system-performance-metric/
Motivation – lookup based

- Scan based methods is popular, but not good enough
  - Contraction Hierarchies (CH)
  - Transit Node Routing (TNR)
  - Customizable Route Planning (CRP)

- Lookup based methods, built for a relational database system
  - Naive way, precompute all pairs result
    - 24M vertices in USA road network
    - At least 6286TB for storage
  - Hub Labeling → HLDB (Microsoft)
  - $\epsilon$-Distance Oracle
  - SILC

- Previous research has shown that lookup based methods achieve a much higher throughput performance
Motivation – distributed key-value store

- High throughput → Distributed system
- Lookup based methods → Key-value data structure → hash access
- We need to adapt distance oracles for distributed architecture
  - Test on RDBMS shows that we cannot scale more than 60K queries/per core
  - Need to scale to even higher throughputs, using Apache Spark

(a) Existing distributed solution

(b) SPDO

Spark Cluster

Loaded in distributed memory
Method - $\epsilon$-Distance Oracle ($\epsilon$-DO)

- A compression representation of road network distances
- Well-separated pair decomposition (WSPD) on a road network
- Can represent the distance between any two points in A and B by one value with an epsilon error tolerance, e.g., 0.25, 0.1, 0.05

$O \left( \frac{n}{\epsilon^2} \right)$ well-separated pairs, each well-separated pair can be represented as a key-value pair

- Key is the pair of two vertex sets, A and B
- Value is $d_G(p_A, p_B)$

*dumbbell structure*
Method - DO-Tree

- Top-down decomposition on the whole space, \((S, S)\)
- PR-Quadtree, Morton code representation

2-dimensional Morton code

\((S, S) \Rightarrow (0, 0)\)

Depth 0

Depth 1

Depth 2

Depth 3

MAX Depth \(D\)

4-dimensional DO-tree

16 branches

\((s_i, t_{j-1}), (s_i, t_j), (s_i, t_{j+1})\)
**Method - DO-Tree**

- **DO-tree** = the top-down WSPD
- Each leaf node of DO-tree is a well-separated pair
- **Uniqueness Property**: For any source-target query \((s, t)\), there is exactly one leaf node of the DO-tree, i.e., WSP that contains both \(s\) and \(t\).
- Task: for any query \((s, t)\), finding the exact one WSP that contains \((s, t)\) **by hash access**, then returning the distance result

---

![Diagram of DO-Tree with nodes and branches](attachment:image.png)
Method - Hash Access for $\epsilon$-DO

- Basic idea (Basic)
  - Hash Table $H_1 = \text{all leaf nodes (WSPs) of DO-tree}$
  - Each query $(s, t)$ generates $O(D)$ keys, each key corresponds one ancestor node of $(s, t)$
  - Exact one key exists in $H_1$ according to the uniqueness property
Method - Hash Access for $\epsilon$-DO

- Binary search method (BS)
  - Hash Table $H_2 = \text{all nodes of DO-tree, } \{\text{root}, (A, B), (C, E)\}$
  - For any query $(s, t)$, initial possible depth range is $[1, D]$
  - Examining if the ancestor node of $(s, t)$ at depth $D/2$ exists in $H_2$ or not
  - Possible depth range becomes either $[0, D/2)$ or $[D/2, D]$
Method - Hash Access for $\varepsilon$-DO

- Binary search method (BS)
  - Hash Table $H_2 = \text{all nodes of DO-tree, \{root, (A, B), (C, E)\}}$
  - Note that $O(\log D) = O(\log \log n)$

- Wise Partitioning method (WP)
  - Adding spatial partitioning of all nodes, push $O(\log D)$ hash lookups in Slaves
## Evaluation – time complexity

- **Time complexity analysis**
  - Master, Slave, and network communications
  - \( N \) source-target queries, \( M \) slaves, \( D \) depths of DO-tree

<table>
<thead>
<tr>
<th>Design</th>
<th>Iteration</th>
<th>Master Time</th>
<th>Slave Time</th>
<th>Network Communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic</td>
<td>1</td>
<td>( O(N \cdot D) )</td>
<td>( O\left(\frac{N \cdot D}{M}\right))</td>
<td>( O(N \cdot D) )</td>
</tr>
<tr>
<td>BS</td>
<td>log ( D )</td>
<td>( O(N \cdot \log D) )</td>
<td>( O\left(\frac{N \cdot \log D}{M}\right))</td>
<td>( O(N \cdot \log D) )</td>
</tr>
<tr>
<td>WP</td>
<td>1</td>
<td>( O(N) )</td>
<td>Random ( O\left(\frac{N \cdot \log D}{M}\right))</td>
<td>( O(N) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Worst ( O(N \cdot \log D) )</td>
<td></td>
</tr>
</tbody>
</table>
Evaluation - throughput

- Comparisons
  - Dijkstra’s algorithm (Dijkstra), good for one-to-many pattern
  - Contraction hierarchies (CH)
  - Distance Oracle embedded in PostgreSQL (DO)

- NYC road network
  - 264K vertices, 733K edges
  - One local server, single-thread

- USA road network
  - 24M vertices, 58M edges
  - 20 machines cluster, single-thread

**Throughput the USA road network**

<table>
<thead>
<tr>
<th>Method</th>
<th>Basic</th>
<th>BS</th>
<th>WP</th>
<th>DO</th>
<th>CH</th>
<th>Dijkstra</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dist/sec/machine</td>
<td>5.0K</td>
<td>25.0K</td>
<td>73.8K</td>
<td>18.8K</td>
<td>385</td>
<td>1.6</td>
</tr>
</tbody>
</table>
Evaluation - applications

- Nearby job opportunities, one-to-many pattern
- Actual drive distance from residence to workplace

One-to-many pattern (2 mins with 1 machine)

One-to-one pattern (13 secs vs. 20 mins in CH with 5 machines)

Nearby job opportunities (e.g., within 10 kms) for each census block in the Bay Area, requiring 120 million distance computations

Average drive distance from residence to workplace for California residents, requiring 13.6 million distance computations
Evaluation - applications

- Route Directness Index (RDI) of two locations
  - Ratio of the network distance to the Euclidean distance

- Define Route Directness Spectrum (RDS) of a spatial region
  - The collection of RDIs between all pairs of vertices in a road network

1 mins per city vs. 4 hours in CH with 1 machine
Conclusions

- SPDO is a high throughput distributed solution for shortest road network distance/time computations using a distributed key-value store on Apache Spark.

- Proposed 3 methods on Apache Spark, Basic, BS, and WP, which are a scalable way of obtaining throughput performance that exceeds one million computations per second with just a few machines.

- Reduce running time for GIS analysts/scientists, save developing time for GIS developers, and simplify the system design and reduce hardware cost for GIS architects.
Thanks