Query Processing in Mobile Peer-to-peer Networks

BY

BO XU
B.S., Beijing University of Aeronautics and Astronautics, 1991
M.S., University of Electronic Science and Technology of China, 1994
Ph.D., University of Electronic Science and Technology of China, 1997

THESIS
Submitted as partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science in the Graduate College of the University of Illinois at Chicago, 2012

Chicago, Illinois

Defense Committee:
Ouri Wolfson, Chair and Advisor
Prasad Sistla, Co-Advisor
Robert Sloan
Peter Scheuermann (Northwestern University)
Philip Yu
This thesis is dedicated to my wife, Ping Xiong, without whom it would never have been accomplished.
ACKNOWLEDGEMENTS

My first debt of gratitude must go to my advisor, Professor Ouri Wolfson. He provided guidance in all areas that helped me accomplish my research goals and enjoy myself in the process. He has been a strong and supportive advisor to me throughout my career as a junior member of academia. I cannot thank him enough. I am forever grateful. Thank You Professor Wolfson!

I would like to give heartfelt thanks to my co-advisor, Professor Prasad Sistla, for his warm encouragement and thoughtful guidance. I feel lucky and honored to have had the chance to work with him. Thanks also to Professors Peter Scheuermann, Robert Sloan, Clement Yu, and Philip Yu for serving in my examination committee and providing valuable comments and suggestions that helped me improve the dissertation.

I would also like to thank my family, coworkers, office staff, and everyone else who helped me through this. Without the guidance and support they provided, this dissertation would not exist.

Finally, I thank the National Science Foundation for their generous support.

BX
# TABLE OF CONTENTS

1 INTRODUCTION ........................................................................................................ 1
1.1 Background ........................................................................................................ 1
1.2 Motivation .......................................................................................................... 4
1.3 Technical Challenges ........................................................................................ 5
1.4 Methodology ...................................................................................................... 6
1.5 Contribution of Work ......................................................................................... 6
   1.5.1 MARKET Query Processing ................................................................. 7
   1.5.2 Application to Real-time Traffic Information ...................................... 8
   1.5.3 Application to Parking Slot Discovery ............................................. 8
   1.5.4 Application to Blobs Queries ............................................................ 9
   1.5.5 Application to Continuous K-nearest-neighbor Queries ............... 9
   1.5.6 Application to Transportation Mode Detection .......................... 10
   1.5.7 Pattern of Reports Propagation ...................................................... 10
   1.5.8 Incentive Mechanisms ..................................................................... 11
   1.5.9 Prototype .............................................................................................. 11
2 LITERATURE REVIEW .......................................................................................... 13
   2.1 Algorithms for Query Processing in Mobile P2P Databases ............ 13
   2.2 Experimental Projects ............................................................................... 16
3 METHODOLOGY ................................................................................................... 18
   3.1 General Model ............................................................................................ 18
   3.2 Ranked Store-and-Forward ...................................................................... 19
4 MARKET QUERY PROCESSING AND ITS APPLICATIONS .......................... 22
   4.1 MARKET Query Processing ..................................................................... 22
   4.1.1 Overview of MARKET ................................................................... 22
   4.1.2 Reports Ranking by Supply and Demand .................................... 24
   4.1.3 Experiments and Results ................................................................ 31
   4.1.4 Relevant Work .................................................................................... 39
   4.2 Application to Parking Slot Discovery ................................................. 40
   4.2.1 Application Environment ................................................................ 40
   4.2.2 Resource-discovery Strategies ....................................................... 42
   4.2.3 Value of Resource Information ...................................................... 43
   4.2.4 Relevant Work .................................................................................... 54
   4.3 Application to Blobs Queries .................................................................... 54
   4.3.1 Application Environment ................................................................ 56
   4.3.2 The WiMaC Query Processing Strategies .................................... 58
   4.3.3 Strategy Dominance Analysis ......................................................... 61
   4.3.4 Comparison of Non-dominated Strategies by Theoretical Analysis 65
   4.3.5 Comparison of Non-dominated Strategies by Simulations .......... 73
   4.3.6 Relevant Work .................................................................................... 88
   4.4 Application to KNN Queries ...................................................................... 88
   4.4.1 KNN Query Processing in MARKET .............................................. 89
   4.4.2 Comparison with DIKNN ................................................................. 89
   4.5 Pattern of Report Propagation .................................................................... 93
   4.5.1 Propagation Pattern of Reports for Temporal Resources .......... 93
   4.5.2 Propagation Pattern of Reports for Spatio-temporal Resources .... 97
   4.5.3 Relevant Work .................................................................................... 100
TABLE OF CONTENTS (continued)

5  CONTINUOUS NEAREST-NEIGHBOR QUERIES WITH LOCATION UNCERTAINTY... 102
   5.1 Contributions and Relevant Work .................................................. 102
   5.1.1 Contributions .............................................................................. 102
   5.2 Continuous kNN with Certain Location ........................................... 112
       5.2.1 Number of Answer-pairs with Linear Motion ............................ 112
       5.2.2 Number of Answer-pairs with Piecewise Linear Motion ............. 119
   5.3 Continuous kNN with Location Uncertainty ..................................... 121
       5.3.1 Uncertainty Model and Continuous Possible kNN Queries .......... 122
       5.3.2 Number of Answer-pairs for CPkNN Queries with Linear Motion . 125
       5.3.3 Number of Answer-pairs for CPkNN Queries with Piecewise Linear Motion  128
   5.4 Query Processing without uncertainty ............................................ 129
       5.4.1 Off-line Processing .................................................................. 129
       5.4.2 On-line Processing .................................................................. 130
   5.5 Query Processing with Uncertainty ................................................. 141
       5.5.1 Off-line Processing .................................................................. 141
       5.5.2 On-line Processing .................................................................. 143
   5.6 Evaluation by experiments ............................................................... 150
       5.6.1 Test Data ............................................................................... 150
       5.6.2 Experimental results ............................................................... 152
   5.7 Summary ....................................................................................... 166

6  TRANSPORTATION MODE DETECTION .............................................. 167
   6.1 Contributions ................................................................. 167
   6.2 Relevant Work ................................................................. 170
   6.3 Preliminaries ................................................................. 172
       6.3.1 Data Model ........................................................................... 172
       6.3.2 General Idea ......................................................................... 173
   6.4 Transportation Mode Detection ..................................................... 174
       6.4.1 System Architecture ............................................................. 174
       6.4.2 Transportation Network Data ................................................ 174
   6.5 Classification Feature Selection ..................................................... 177
       6.5.1 Average accuracy of GPS coordinates .................................... 177
       6.5.2 Average speed ....................................................................... 178
       6.5.3 Average heading change ....................................................... 178
       6.5.4 Average acceleration ............................................................. 179
       6.5.5 Bus location closeness .......................................................... 179
       6.5.6 Rail line trajectory closeness .................................................. 180
       6.5.7 Bus stop closeness rate .......................................................... 181
       6.5.8 Zip code based Indexing and Pruning .................................... 182
   6.6 Evaluation .................................................................................. 183
       6.6.1 Data Collection ..................................................................... 183
       6.6.2 Training data preprocessing .................................................. 184
       6.6.3 Window Size ......................................................................... 185
       6.6.4 Classifier selection ............................................................... 185
       6.6.5 Mode detection accuracy ....................................................... 186
       6.6.6 Performance and scalability ................................................... 192
       6.6.7 Extended real world evaluation ............................................ 193
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.6.8 Bus mode detection discussion</td>
<td>195</td>
</tr>
<tr>
<td>6.7 Summary</td>
<td>196</td>
</tr>
<tr>
<td>7 CONCLUSION AND FUTURE WORK</td>
<td>198</td>
</tr>
<tr>
<td>CITED LITERATURE</td>
<td>202</td>
</tr>
<tr>
<td>APPENDICES</td>
<td>212</td>
</tr>
<tr>
<td>CURRICULUM VITAE</td>
<td>217</td>
</tr>
<tr>
<td>FIGURE</td>
<td>PAGE</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Mobile ad hoc solution to local search</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>A mobile P2P database</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Mobile P2P database augments the infrastructure</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>Query processing methods in mobile P2P databases</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td>An example of the ranked store-and-forward procedure</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>24</td>
</tr>
<tr>
<td>Flow chart of the MARKET algorithm at a peer</td>
<td>24</td>
</tr>
<tr>
<td>7</td>
<td>29</td>
</tr>
<tr>
<td>Pseudo-code of MALENA algorithm executed at peer O</td>
<td>29</td>
</tr>
<tr>
<td>8</td>
<td>31</td>
</tr>
<tr>
<td>Detail steps of Query-response interaction executed at peer A when A encounters a peer B</td>
<td>31</td>
</tr>
<tr>
<td>9</td>
<td>36</td>
</tr>
<tr>
<td>Response-time bounded throughput of compared algorithms</td>
<td>36</td>
</tr>
<tr>
<td>10</td>
<td>38</td>
</tr>
<tr>
<td>Throughput as a function of RDB size</td>
<td>38</td>
</tr>
<tr>
<td>11</td>
<td>38</td>
</tr>
<tr>
<td>Throughput as a function of Transmission size</td>
<td>38</td>
</tr>
<tr>
<td>12</td>
<td>44</td>
</tr>
<tr>
<td>The grid network, resources, search square, brokers and consumers</td>
<td>44</td>
</tr>
<tr>
<td>13</td>
<td>49</td>
</tr>
<tr>
<td>Discovery time versus broadcast size</td>
<td>49</td>
</tr>
<tr>
<td>14</td>
<td>49</td>
</tr>
<tr>
<td>Discovery time versus IGS consumer ratio</td>
<td>49</td>
</tr>
<tr>
<td>15</td>
<td>50</td>
</tr>
<tr>
<td>Discovery time versus number of consumers</td>
<td>50</td>
</tr>
<tr>
<td>16</td>
<td>50</td>
</tr>
<tr>
<td>Discovery time versus mean of invalid duration</td>
<td>50</td>
</tr>
<tr>
<td>17</td>
<td>51</td>
</tr>
<tr>
<td>Discovery time versus mean of motion speed</td>
<td>51</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>18</td>
<td>Discovery time versus broker density</td>
</tr>
<tr>
<td>19</td>
<td>Discovery time versus bandwidth allocation</td>
</tr>
<tr>
<td>20</td>
<td>Dominance relationship among strategies</td>
</tr>
<tr>
<td>21</td>
<td>Illustration of the 1 (blob) strategy</td>
</tr>
<tr>
<td>22</td>
<td>Illustration of the 3a (Q)-WiFi strategy</td>
</tr>
<tr>
<td>23</td>
<td>Illustration of the 6b (meta,Q)-cell strategy</td>
</tr>
<tr>
<td>24</td>
<td>Delivery ratio versus encounter intensity, with N+1=100 and T=300 seconds</td>
</tr>
<tr>
<td>25</td>
<td>Delivery ratio versus query lifetime, with N+1=100 and $\lambda=0.001$ sec$^{-1}$</td>
</tr>
<tr>
<td>26</td>
<td>Delivery ratio versus number of peers, with $\lambda=0.0005$ sec$^{-1}$ and T=300 seconds</td>
</tr>
<tr>
<td>27</td>
<td>Simulated road network: portion of the highway system in a US city</td>
</tr>
<tr>
<td>28</td>
<td>Answer throughput versus penetration ratio, light-congestion, SWANS++, report lifetime = 300 seconds, query ratio = 0.25</td>
</tr>
<tr>
<td>29</td>
<td>Answer throughput versus penetration ratio, heavy-congestion, SWANS++, report lifetime = 300 seconds, query ratio = 0.25</td>
</tr>
<tr>
<td>30</td>
<td>Answer throughput versus report lifetime, light-congestion, SWANS++, penetration ratio = 0.05, query ratio = 0.25</td>
</tr>
<tr>
<td>31</td>
<td>Answer throughput versus report lifetime, heavy-congestion, SWANS++, penetration ratio = 0.05, query ratio = 0.25</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>32</td>
<td>Answer throughput versus query ratio, light-congestion, SWANS++, penetration ratio = 0.05, report lifetime = 300 seconds</td>
</tr>
<tr>
<td>33</td>
<td>Answer throughput versus query ratio, light-congestion, SWANS++, penetration ratio = 0.05, report lifetime = 300 seconds</td>
</tr>
<tr>
<td>34</td>
<td>Communication overhead versus penetration ratio, heavy-congestion, SWANS++, penetration ratio = 25%, report lifetime = 300 seconds, query ratio = 0.25</td>
</tr>
<tr>
<td>35</td>
<td>Answer throughput versus penetration ratio, light-congestion, simple, report lifetime = 300 seconds, query ratio = 0.25</td>
</tr>
<tr>
<td>36</td>
<td>Answer throughput versus penetration ratio, heavy-congestion, simple, report lifetime = 300 seconds, query ratio = 0.25</td>
</tr>
<tr>
<td>37</td>
<td>Accuracy as a function of peer density with k=10</td>
</tr>
<tr>
<td>38</td>
<td>Accuracy as a function of k with peer density = 2</td>
</tr>
<tr>
<td>39</td>
<td>Accuracy as a function of k with peer density = 1. The accuracy is zero for DIKNN for all the k values</td>
</tr>
<tr>
<td>40</td>
<td>Number of copies of a report as a function of its age</td>
</tr>
<tr>
<td>41</td>
<td>Propagation pattern of reports</td>
</tr>
<tr>
<td>42</td>
<td>Example of answer-pairs in the Time-SquareDistance space</td>
</tr>
<tr>
<td>43</td>
<td>The auxiliary figure for the proof of Theorem 5.1</td>
</tr>
<tr>
<td>44</td>
<td>The auxiliary figure for the proof of Proposition 5.1</td>
</tr>
</tbody>
</table>
LIST OF FIGURES (continued)

45. In the piecewise linear model each object is represented by a connected sequence of parabola-pieces

46. Uncertainty regions, minimum possible distance, and maximum possible distance

47. Max-curves (solid parabolas) and min-curves (dashed parabolas)

48. The auxiliary figure for the proof of Theorem 5.3

49. Kinetic tournament

50. The object heap at time 0 for the arrangement of Figure 49(a)

51. The object heap resulting from an implicit update on the object heap of Figure 50

52. The object heap resulting from an explicit update on the object heap of Figure 50

53. Case 2 of addition and Case 2 of deletion

54. \(r.time\) may change due to the velocity-vector change of an object other than \(r.object\)

55. Query processing time versus number of objects

56. Query processing time versus \(k\)

57. Comparison between on-line and off-line as a function of the number of objects when there are no updates in the certain case

58. Comparison between on-line and off-line as a function of the total number of
LIST OF FIGURES (continued)

velocity-vector updates for all the objects in the certain case

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>59</td>
<td>Comparison between on-line and off-line as a function of the total number of velocity-vector updates for all the object in the uncertain case</td>
<td>156</td>
</tr>
<tr>
<td>60</td>
<td>Number of answer-pairs versus number of objects</td>
<td>158</td>
</tr>
<tr>
<td>61</td>
<td>Number of answer-pairs versus $k$</td>
<td>158</td>
</tr>
<tr>
<td>62</td>
<td>Number of answer-pairs versus uncertainty region radius</td>
<td>159</td>
</tr>
<tr>
<td>63</td>
<td>Size of PkNN set versus number of objects</td>
<td>159</td>
</tr>
<tr>
<td>64</td>
<td>Size of PkNN set versus $k$</td>
<td>160</td>
</tr>
<tr>
<td>65</td>
<td>Size of PkNN set versus uncertainty region radius</td>
<td>160</td>
</tr>
<tr>
<td>66</td>
<td>Query processing time versus number of objects</td>
<td>163</td>
</tr>
<tr>
<td>67</td>
<td>Query processing time versus $k$</td>
<td>163</td>
</tr>
<tr>
<td>68</td>
<td>Query processing time versus uncertainty region radius</td>
<td>164</td>
</tr>
<tr>
<td>69</td>
<td>Space overhead of object heap versus number of objects</td>
<td>164</td>
</tr>
<tr>
<td>70</td>
<td>Related work with GPS sensor</td>
<td>172</td>
</tr>
<tr>
<td>71</td>
<td>Generating classification examples from GPS sensor and transportation network data</td>
<td>174</td>
</tr>
<tr>
<td>72</td>
<td>Real-time bus locations</td>
<td>176</td>
</tr>
<tr>
<td>73</td>
<td>Rail network</td>
<td>177</td>
</tr>
<tr>
<td>74</td>
<td>Mobile user’ Euclidian distance to closest bus stop while riding a bus</td>
<td>182</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>75</td>
<td>Devices used for data collection (HP IPAQ, Samsung Galaxy, IPhone 3G)</td>
<td>184</td>
</tr>
<tr>
<td>76</td>
<td>Data collection user interface</td>
<td>185</td>
</tr>
<tr>
<td>77</td>
<td>Transportation network features not considered</td>
<td>188</td>
</tr>
<tr>
<td>78</td>
<td>Transportation network features considered</td>
<td>189</td>
</tr>
<tr>
<td>79</td>
<td>Classification feature ranking and selection</td>
<td>191</td>
</tr>
<tr>
<td>80</td>
<td>Only five high order classification features used.</td>
<td>192</td>
</tr>
<tr>
<td>81</td>
<td>Deployed classification system</td>
<td>194</td>
</tr>
<tr>
<td>82</td>
<td>Evaluation of deployed system</td>
<td>195</td>
</tr>
<tr>
<td>83</td>
<td>Effects of CBC on bus mode detection</td>
<td>196</td>
</tr>
<tr>
<td>84</td>
<td>The auxiliary figure for the proof of Proposition 5.2.</td>
<td>212</td>
</tr>
<tr>
<td>85</td>
<td>The Concrete Example for the Proof of Theorem 5.5</td>
<td>215</td>
</tr>
<tr>
<td>86</td>
<td>Proposition 5.3 does not hold when there is location uncertainty associated with the query object</td>
<td>126</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>TABLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>16</td>
</tr>
<tr>
<td>II</td>
<td>17</td>
</tr>
<tr>
<td>III</td>
<td>34</td>
</tr>
<tr>
<td>IV</td>
<td>47</td>
</tr>
<tr>
<td>V</td>
<td>60</td>
</tr>
<tr>
<td>VI</td>
<td>70</td>
</tr>
<tr>
<td>VII</td>
<td>78</td>
</tr>
<tr>
<td>VIII</td>
<td>79</td>
</tr>
<tr>
<td>IX</td>
<td>91</td>
</tr>
<tr>
<td>X</td>
<td>99</td>
</tr>
<tr>
<td>XI</td>
<td>108</td>
</tr>
<tr>
<td>XII</td>
<td>151</td>
</tr>
<tr>
<td>XIII</td>
<td>153</td>
</tr>
<tr>
<td>XIV</td>
<td>162</td>
</tr>
<tr>
<td>XV</td>
<td>162</td>
</tr>
<tr>
<td>XVI</td>
<td>165</td>
</tr>
</tbody>
</table>

I Pedestrians mobile peer-to-peer projects
II Vehicular mobile peer-to-peer projects
III Simulation parameters and their values
IV All parameters and their values
V Design space of the WiMaC paradigm
VI System parameters and their values
VII Simulation parameters and their values
VIII Mapping between the penetration ratio and the average distance between two neighboring participating vehicles
IX Simulation parameters and their values
X Simulation parameters and their values
XI Summary of the results
XII Experimental parameters and their values.
XIII Cost components when processing a velocity-vector update
XIV Processing time distribution for the uncertain case when \( N=10000, k=1, \) and \( r=15\text{m}. \)
XV Processing time distribution for the certain case when \( N=1000000 \) and \( k=1 \)
XVI Average length of the event queue in the uncertain case when \( k=1 \) and
LIST OF TABLES (continued)

r=15m
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC</td>
<td>Average Bus Closeness</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ARLC</td>
<td>Average Rail Location Closeness</td>
</tr>
<tr>
<td>BLOB</td>
<td>Binary Large OBjects</td>
</tr>
<tr>
<td>BN</td>
<td>Bayesian Network</td>
</tr>
<tr>
<td>BRR</td>
<td>Blob-Reports Relation</td>
</tr>
<tr>
<td>BS</td>
<td>Blind Search</td>
</tr>
<tr>
<td>BSCR</td>
<td>Bus Stop Closeness Rate</td>
</tr>
<tr>
<td>CBC</td>
<td>Candidate Bus Closeness</td>
</tr>
<tr>
<td>CkNN</td>
<td>Continuous $k$ Nearest Neighbors</td>
</tr>
<tr>
<td>CPkNN</td>
<td>Continuous Possible $k$ Nearest Neighbors</td>
</tr>
<tr>
<td>CPT</td>
<td>Conditional Probability Tables</td>
</tr>
<tr>
<td>CRAWDAD</td>
<td>Community Resource for Archiving Wireless Data At Dartmouth</td>
</tr>
<tr>
<td>CTA</td>
<td>Chicago Transit Authority</td>
</tr>
<tr>
<td>DIKNN</td>
<td>Density-aware Itinerary KNN query processing</td>
</tr>
<tr>
<td>DT</td>
<td>Decision Trees</td>
</tr>
<tr>
<td>EGNOS</td>
<td>European Geostationary Navigation Overlay Service</td>
</tr>
<tr>
<td>FIFO</td>
<td>First-In-First-Out</td>
</tr>
<tr>
<td>GAF</td>
<td>Geographic Adaptive Fidelity</td>
</tr>
<tr>
<td>GEAR</td>
<td>Geographical and Energy Aware Routing</td>
</tr>
<tr>
<td>GIS</td>
<td>Geographical Information System</td>
</tr>
<tr>
<td>GPS</td>
<td>Global Positioning System</td>
</tr>
<tr>
<td>GPSR</td>
<td>Greedy Perimeter Stateless Routing</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Full Form</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
</tr>
<tr>
<td>GRS</td>
<td>Greedy Reports Selection</td>
</tr>
<tr>
<td>GSM</td>
<td>Global System for Mobile Communications</td>
</tr>
<tr>
<td>IGS</td>
<td>Information Guided Search</td>
</tr>
<tr>
<td>KNN</td>
<td>K Nearest Neighbor</td>
</tr>
<tr>
<td>LFU</td>
<td>Least Frequently Used</td>
</tr>
<tr>
<td>LRU</td>
<td>Least Recent Used</td>
</tr>
<tr>
<td>MAC</td>
<td>Medium Access Control</td>
</tr>
<tr>
<td>MALENA</td>
<td>MAchine LEarning-based Novelty rAnking</td>
</tr>
<tr>
<td>MANET</td>
<td>Mobile Ad-hoc NETwork</td>
</tr>
<tr>
<td>ML</td>
<td>Multiplayer Perceptron</td>
</tr>
<tr>
<td>MP2P</td>
<td>Mobile Peer-to-Peer</td>
</tr>
<tr>
<td>MP2PD</td>
<td>Mobile Peer-to-Peer Database</td>
</tr>
<tr>
<td>MRR</td>
<td>Metadata-Reports Relation</td>
</tr>
<tr>
<td>NB</td>
<td>Naïve Bayes</td>
</tr>
<tr>
<td>NI</td>
<td>Number Intervals</td>
</tr>
<tr>
<td>P2P</td>
<td>Peer-to-Peer</td>
</tr>
<tr>
<td>PDA</td>
<td>Personal Digital Assitant</td>
</tr>
<tr>
<td>PkNN</td>
<td>Possible k Nearest Neighbor</td>
</tr>
<tr>
<td>QDB</td>
<td>Queries Database</td>
</tr>
<tr>
<td>QRR</td>
<td>Query-Reports Relation</td>
</tr>
<tr>
<td>RANDI</td>
<td>RANk-based DIssemination</td>
</tr>
<tr>
<td>Ranked-SF</td>
<td>Ranked Store-and-Forward</td>
</tr>
<tr>
<td>RDB</td>
<td>Reports Database</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Framework</td>
</tr>
<tr>
<td>RFID</td>
<td>Radio-frequency identification</td>
</tr>
<tr>
<td>RFID</td>
<td>Random Forest</td>
</tr>
<tr>
<td>RS</td>
<td>Reports Selection</td>
</tr>
<tr>
<td>SWANS</td>
<td>Scalable Wireless Ad-hoc Network Simulator</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>UWB</td>
<td>Ultra Wide Band</td>
</tr>
<tr>
<td>VANET</td>
<td>Vehicular Ad-hoc Network</td>
</tr>
<tr>
<td>VII</td>
<td>Vehicle Infrastructure Integration</td>
</tr>
<tr>
<td>WAAS</td>
<td>Wide Area Augmentation System</td>
</tr>
<tr>
<td>WCM</td>
<td>WiFi Communication Module</td>
</tr>
<tr>
<td>WiMaC</td>
<td>WiFi-communication, Match, Communication</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
</tbody>
</table>
SUMMARY

Mobile local search is a procedure in which a mobile user searches for local resources, i.e., resources that are in geographic proximity to the user (e.g., a person with certain expertise in a convention hall, a ride-share opportunity, a taxi-cab, a parking slot, etc.). A promising approach to mobile local search is mobile peer-to-peer databases (MP2PD). In the MP2PD approach, a database is stored in the peers (PDA's, cell phones, vehicles, sensors, etc.) that communicate with each other via short-range wireless technologies such as IEEE 802.11, Bluetooth, Zigbee, or Ultra Wide Band (UWB). All the local databases maintained by the mobile peers form a mobile P2P database. The characteristics of MP2PD include (i) dynamic network topology, (ii) memory/energy/bandwidth limitations, and (iii) lack of global coordination. The objective of the dissertation is to study query processing in such an environment.

The traditional in-network query processing paradigm postulates that a query is routed among peers and collects the answers from the peers. It works for static and connected networks. However, when the network consists of mobile peers and is sparse, a different approach is necessary. We propose a query processing method that uses cooperative caching. It makes the data items satisfying a query flow to its originator. To cope with communication bandwidth and storage constraints, the method prioritizes the data-items in terms of their value, as reflected by supply and demand. The dissertation develops the formula by which a mobile peer dynamically adjusts the number of reports included in a transmission, develops a report prioritization method called MARKET, analyzes the way a report is propagated in geospace and time, the benefit of information dissemination in capturing competitive resources (i.e. resources that can be used by only one user at a time) and provisioning real-time traffic information, how well the average local database reflects the status of physical resources, how our approach compares with the central server model and with existing work on publish/subscribe in wireless ad-hoc networks, how to incentivize mobile devices to relay reports, the application to continuous $k$NN queries and to transportation mode detection.
1 INTRODUCTION

In this chapter we provide background and motivation for the research. We also present the technical challenges and summarize the contributions of the dissertation.

1.1 Background

Mobile local search is a procedure in which a mobile user searches for local resources, i.e. resources that are in geographic proximity to the user (e.g., a person with certain expertise in a convention hall, a ride-share opportunity, a taxi-cab, a parking slot, etc). We will argue that mobile local search is more effectively and efficiently conducted in a peer-to-peer paradigm. In this paradigm, a database is stored in the peers¹ (PDA’s, cell phones, vehicles, sensors, etc) of a Mobile Ad-hoc NETwork (MANET) (Figure 1). These peers communicate with each other via broadband (typically tens of Mbps) but short-range wireless technologies such as IEEE 802.11, Bluetooth, Zigbee [1], or Ultra Wide Band (UWB) [2]. On each mobile peer there is a local database that stores and manages a collection of reports, where each report describes a resource. Thus a report is analogous to a web page in Google search. All the local databases maintained by the mobile peers form a mobile peer-to-peer database (MP2PD). The peers communicate reports and queries (a query is analogous to a search in Google terminology) to neighbors directly, and the reports and queries propagate by transitive multi-hop transmissions. In contrast to traditional information management systems, an infrastructure and a central server may not exist, and the information hosts are autonomous and mobile. Figure 2 below illustrates the concept of a mobile P2P database.

¹ Peers are equal in terms of the communication and the data management protocol they use. But they are not necessarily the same device type nor have the same capabilities.
The MP2PD does not require a central server or a wireless infrastructure. However, when the infrastructure is available, the MP2PD can augment it to make the search more efficient. The cellular and mobile P2P approaches can be combined into an architecture in which resource-information disseminated in a mobile P2P network augments the infrastructure by covering the areas that are not covered by the infrastructure (e.g. elevators, subways, disaster areas), and it enhances and lowers the cost of local search where offered by the infrastructure. In other words, the P2P approach can also be used to communicate among the leaves of a hierarchical cellular architecture (see Figure 3), further enhancing the search.
capability. In Figure 3, rectangles are access points of a possibly fixed hierarchical infrastructure, each of which controls an area called a “cell”. For example, the rectangles may represent cellular towers.

Figure 3. Mobile P2P database augments the infrastructure

MP2PD has applications in many domains, including social networks, transportation, mobile electronic commerce, emergency response, homeland security. For example, in a large professional, political, or social gathering, the technology can be used to automatically facilitate a face-to-face meeting based on matching interest profiles which circulate at the venue. In transportation, the platform incorporated in navigational devices can be used to disseminate to other similarly-equipped vehicles information about relevant resources such as free parking slots, traffic jams and slowdowns, available taxicabs, and ride sharing. In mobile electronic commerce, the technology is useful to match buyers and sellers in a mall, or to disseminate information about a marketed product to passers-by. In disaster recovery, MP2PD software can be used by first responders to support rescue efforts when the fixed infrastructure is inoperative. In homeland security, sensors on containers can transitively relay alerts to remote check-points.
1.2 **Motivation**

Consider mobile users that search for local resources. Assuming that the information about the existence and location of such a resource resides on a server, a communication infrastructure is necessary to access the server. Such an infrastructure may not be available in military/combat situations, disaster recovery, in a commercial flight, etc. Even if the infrastructure and a server are both available, a user may not be willing to pay the dollar-cost that is usually involved in accessing the server through the cellular infrastructure. Furthermore, cellular bandwidth is limited (e.g. 130 character text messages). In other words, a client-server approach may have accessibility problems.

Currently, Google and local.com provide static local information (e.g. the location of a restaurant, pharmacy, etc.), but not dynamic information such as the location of a taxi cab, a nearby person of interest, or an available parking slot. These dynamic resources are temporary in nature, and thus require timely, real-time update rates. Such rates are unlikely to be provided for the country or the world by a centralized server farm, a la Google. Thus, dynamic local resources may require local servers, each dedicated to a limited geographic area. However, for many areas such a local server may not exist due to lack of a profitable business model, and if it exists it may be unavailable (such servers are unlikely to have the reliability of global sites such as Google). Furthermore, the data on the server may be unavailable due to propagation delays (think of sudden-brake information that needs to be propagated to a server and from there to the trailing vehicles), or due to device limitations (e.g. a cab customer’s cell-phone may have Bluetooth but not internet access to update the server), or due to the fact that updates from mobile devices may involve a communication cost that nobody is willing to pay, or due to the fact that the local server (e.g. of Starbucks) may accept only updates from certain users or certain applications but not others. In short, a client-(local)-server may have both accessibility and availability problems.

We propose to substitute or augment the client-(local)-server approach by a MP2PD. The MP2PD is distributed among the mobile peers. Communication in the mobile P2P network is free since it uses the unlicensed spectrum, and larger in bandwidth than the cellular infrastructure, thus can provide media rich
information, such as maps, menus, and even video. A mobile user may search the MP2PD only, or combine it with a client-server search.

From the historical point of view, the currently prevalent centralized architecture was designed at a time when the mobile devices did not possess the capability to make them intelligent. And therefore, in such an architecture, the intelligence of mobile communication lies in the fixed network infrastructure, and not in the devices. However, nowadays, mobile devices have hundreds-of-MHz processors, and tens of megabytes of random access memory. So another way of looking at this dissertation is as a shift of paradigm towards intelligent devices.

1.3 Technical Challenges

**Mobility and sparseness of peers.** In our environment, the participating parties are physically mobile (sometimes highly so, e.g., consider vehicles that travel in opposite directions at 120 miles/hour relative speed). The peer density can vary in a big range from a crowded convention center to a midnight highway. The underlying communication network is thus subject to topology changes and disconnections. In such an environment a mobile peer does not necessarily always have neighboring peers to communicate with, and even if it does, the set of the neighbors is not fixed. Furthermore, due to insufficient density of peers, there may not exist a contemporaneous communication path between a pair of peers. These characteristics defeat the applicability of typical peer-to-peer frameworks that rely on data access structures and routing schemes (e.g. Gnutella [3], DHT’s like [4], and Gridella [5]). Furthermore, existing approaches (Mobile Ad Hoc Networks (MANET's) [6], and Mesh Networks [7]) do not address search (but message routing from one peer to another).

**Bandwidth, energy, and memory constraints on mobile peers.** Communication between mobile peers is often restricted by bandwidth and energy constraints. Furthermore, often reports need to be stored and later forwarded, thus memory constraints on the mobile peers constitute a problem as well. Thus, careful and efficient utilization of scarce peer resources (specifically bandwidth, power, and memory) are an important challenge.
1.4 Methodology

We adopt a store-and-forward paradigm which exploits mobility to cope with disconnections. Specifically, on each mobile peer there is a local database that stores and manages a collection of data items, or reports. A report is a set of attribute-values sensed (e.g., with an RFID reader, a camera, etc.) or received by a peer at a particular time. For example, for the KNN query processing, a report gives the location (sensed by a GPS receiver) of a mobile peer at a particular time. All the local databases maintained by the mobile peers form the mobile P2P database. A peer answers its query from its local database. For query processing, the peers communicate reports and queries to neighbors directly, and the reports and queries propagate by transitive multi-hop transmissions. The intermediate peers (also called brokers) need to save reports and later, as new neighbors are discovered, transfer these reports. Thus each mobile peer in the network is a broker, and additionally it may be a consumer or a producer of reports, or both. In terms of functionality, we envision peers as equals (i.e., no “super-peers”). Since store-and-forward does not need to build and maintain any routing structure, it withstands highly dynamic topologies.

The problem with store-and-forward algorithms is that the reports that need to be stored and forwarded by a peer may exceed its storage, bandwidth capacities. We propose to address this problem by ranking of reports, so that the most relevant reports are transmitted and saved to the local database. We call this approach ranked store-and-forward (Ranked-SF).

1.5 Contribution of Work

In our recent work [8-36], we explored the Ranked-SF approach. More specifically, we developed the formula by which a mobile peer dynamically adjusts the number of reports included in a transmission ([21]), developed a report prioritization method called MARKET ([11, 16, 19, 32]), analyzed the way a report is propagated in geospace and time ([29]), the benefit of information dissemination in capturing competitive resources (i.e. resources that can be used by only one user at a time) (see [8, 9, 10, 14]) and provisioning real-time traffic information ([26, 28, 31]), how well the average local database reflects the
status of physical resources ([18]), how our approach compares with the central server model and with existing work on publish/subscribe in wireless ad-hoc networks (see [11, 19, 23, 32]), how to incentivize mobile devices to relay reports (see [12, 13, 14]), the application to continuous kNN queries ([34, 35]) and the application to transportation mode detection ([33]). We also built an initial prototype of a MP2P data dissemination system ([36]).

1.5.1 MARKET Query Processing

In the MARKET algorithm, the growing-local-database problem of store-and-forward algorithms is addressed by prioritization; each mobile peer prioritizes the reports in order to accommodate them in limited power, bandwidth, and memory. The priority of a report depends on its size (the larger the report, the more resources it consumes), demand (how many peers are querying it), and supply (how many peers already have it). The demand of a report is estimated by sampling of the queries that it satisfies. But sampling does not work for estimating the supply, because supply increases continuously as the report is being disseminated. Thus, we developed and implemented an algorithm called MALENA that combines various indicators to estimate the current supply [24]. The combination uses a machine learning system that is trained from previously received reports, and automatically learns the best indicator for the current environment.

An additional issue arising in a store-and-forward algorithm such as MARKET is how many reports to communicate in each transmission. If too many, excessive collisions arise, and if too few, then the search capability suffers. We developed an analytical model that computes the throughput of a transmission in an 802.11 ad hoc network (see [21]). The throughput is computed based on collision factors such as the transmission size, the transmission frequency, the density of mobile peers, and so on. Using this analytical model we proposed a method by which a mobile peer dynamically adjusts the P2P transmission size depending on the period of time between two P2P transmissions of a peer. The objective is to optimize the bandwidth and energy utilization; optimization occurs when the number of reliably received reports delivered per unit of energy is maximized. Simulations show that by dynamically
adjusting the transmission size, the performance of the mobile P2P search is improved by up to an order of magnitude.

We compared MARKET with existing mobile P2P algorithms, including PeopleNet [37], and 7DS [38], LRU, and LFU. MARKET outperforms PeopleNet and 7DS by an order of magnitude. It outperforms LRU when the database size is small and the transmission size is big. More details are provided in [11, 32].

1.5.2 Application to Real-time Traffic Information

We studied the application of MARKET in the dissemination of real-time traffic information. In this application, the real-time traffic information is produced by and disseminated to vehicles throughout a road network. This application complements the existing real-time traffic information dissemination methods which tend to cover only selected highways where speed sensors are deployed. We compared MARKET with Grassroots [39], a flooding based mobile P2P traffic information dissemination algorithm. The results show that MARKET outperforms Grassroots when the vehicle density is sparse or when the available bandwidth is small. In some cases MARKET outperforms Grassroots by 50%. These results demonstrate the benefit of store-and-forward, information prioritization, and bandwidth adaptation. More details are provided in [26, 28, 31].

1.5.3 Application to Parking Slot Discovery

We considered discovery of parking slots in a mobile P2P network, and conducted a comparative study of alternatives. One contribution is to quantify the benefit of using resource-discovery information. We defined this benefit to be the (resource-discovery-time-without-information) minus (resource-discovery-time-with-information). As far as we know, this benefit has not been quantified previously. Our experiments show that by using resource-discovery information disseminated via MP2PD the resource discovery time can be cut by as much as 70%. More details are provided in [8, 9, 10, 14].
1.5.4 **Application to Blobs Queries**

We studied querying binary large objects such as video and voice clips in mobile peer-to-peer networks. A mobile peer-to-peer network consists of mobile peers that are capable of both infrastructure-less short-range communication and infrastructure communication. We introduced a set of query processing strategies which differ from each other in terms of push versus pull, whether or not infrastructure communication is utilized, and whether metadata dissemination is separated from blob dissemination. We analyzed these strategies theoretically and by simulations, and identify the one that is superior to the others. Preliminary work is reported in [30].

1.5.5 **Application to Continuous K-nearest-neighbor Queries**

We demonstrated the MARKET algorithm in the processing of continuous K-nearest-neighbor queries. In this case, the reports are the current locations and velocity-vectors of mobile sensors. Thus, the MARKET algorithm is specialized to process this specific spatial query in a data-to-query fashion. Similarly, it can be specialized for in-network processing of other types of queries, e.g. spatial window queries.

We studied the problem of evaluating the continuous query of finding the $k$ nearest peers with respect to a given mobile peer $O_q$ among a set of $n$ mobile peers. The query returns a sequence of answer-pairs, namely pairs of the form $(I, S)$ such that $I$ is a time interval and $S$ is the set of peers that are closest to $O_q$ during $I$. When there is uncertainty associated with the locations of the mobile peers, $S$ is the set of all the peers that are possibly the $k$ nearest neighbors. We analyzed the lower bound and the upper bound on the maximum number of answer-pairs, for the certain case and the uncertain case, respectively. Then we considered two different types of algorithms. The first is off-line algorithms that compute a priori all the answer-pairs. The second type is on-line algorithms that at any time return the current answer-pair. We developed algorithms for the certain case and the uncertain case respectively and analyzed their complexity. We experimentally compared different algorithms using a database of 1 million peers derived from real-world GPS traces.
We compared the MARKET KNN performance with that of the in-network KNN algorithm given in [40], called DIKNN. It processes queries in a query-to-data fashion. The results show that MARKET is up to 50 times more accurate than DIKNN when the P2P network is sparse, but DIKNN is more accurate when the network is dense. More details are provided in [32].

1.5.6 Application to Transportation Mode Detection

The transportation mode such as walking, cycling or on a train denotes an important characteristic of the mobile user’s context. We proposed an approach to inferring a user’s mode of transportation based on the GPS sensor on her mobile device and knowledge of the underlying transportation network. The transportation network information considered includes real time bus locations, spatial rail and spatial bus stop information. This information can be exchanged among mobile peers by peer-to-peer communication. We identified and derived the relevant features related to transportation network information to improve classification effectiveness. This approach achieved over 93.5% accuracy for inferring various transportation modes including: car, bus, aboveground train, walking, bike, and stationary. Our approach improved the accuracy of detection by 17% in comparison with the GPS only approach, and 9% in comparison with GPS with GIS models. Additionally, if a user is travelling by bus, our approach is able to provide further information about which particular bus the user is riding. Five different inference models including Bayesian Net, Decision Tree, Random Forest, Naïve Bayesian and Multilayer Perceptron, were tested in the experiments. The final classification system was deployed and made available to the public.

1.5.7 Pattern of Reports Propagation

We studied how a report is propagated with the MARKET algorithm. Such a pattern shows that, by very simple local decisions made at each vehicle, the opportunistic dissemination algorithm automatically limits the global distribution of a resource to a bounded spatial area, which is a circle around the home location of the resource. The algorithm also limits the distribution to the time-duration for which the
resource is of interest. Furthermore, the spatial and temporal boundaries automatically adapt depending on the number of resources in the system, the traffic density and speed, and other parameters that dictate the amount of storage, processing power, and bandwidth that should be allocated to each resource. For example, if the number of resources is small, each resource will stay in the system longer, and spread farther. More details are provided in [19].

1.5.8 Incentive Mechanisms

We developed various incentive mechanisms to stimulate cooperation in an opportunistic peer-to-peer environment. The incentive mechanisms are based upon virtual currency. Each mobile node carries virtual currency in the form of a counter that is protected from illegitimate manipulation by a trusted and tamper resistant hardware module. The incentive mechanisms differ in who pays, who charges, and how much is paid or charged. We experimentally showed that paying for the propagation of reports is well justified by the benefit gained by the payer, which in turn justifies the incentive models. Finally, we quantify by simulations the best tradeoff between the payment and the benefit. More details are provided in [12, 13, 14].

1.5.9 Prototype

We have implemented the communication layer on the Pocket PC/WinCE PDA platform. It allows mobile PDA’s to conduct multi-hop WiFi peer-to-peer communication without any base station. We have built a generic GUI generator that allows an application developer to define the schema of reports and queries for a given application. The generator then automatically creates forms for the end users to access the mobile local information dissemination system, including issuing queries and viewing matched reports. We have also implemented several simple applications upon the communication layer, demonstrating ride-sharing and discovery of cab customers. Finally, we have implemented an application in which PDA’s mounted in vehicles’ dashboards capture traffic video clips; the video clips are disseminated and received by nodes in the VANET. A video of the experiment is available (see [36]).
The rest of this dissertation is organized as follows. Chapter 2 provides a detailed review of the relevant work on the topic. Chapter 3 explains in detail the research methodology that will be used in this study, and how it will be executed. Chapter 4 presents MP2P query processing and its application to parking slot discovery and blobs queries. Chapter 5 discusses the application of MP2P to continuous k-nearest-neighbor queries. Chapter 6 discusses the application of MP2P to transportation mode detection. Chapter 7 concludes the dissertation and discusses future work.
2 LITERATURE REVIEW

In this chapter we provide a detailed review of the relevant work on query processing in mobile P2P databases. We discuss relevant work from the following two aspects:

- Algorithms for query processing in mobile P2P databases.
- Experimental projects

2.1 Algorithms for Query Processing in Mobile P2P Databases

There are two main paradigms for query processing in mobile P2P databases, one is report pulling and the other one is report pushing.

Report pulling means that a mobile peer issues a query which is flooded in the whole network, and the answer-reports will be pulled from the mobile peers that have them (see e.g., [41]). Report pulling is widely used in resource discovery, such as route discovery in mobile ad hoc networks and file discovery by query flooding in wired P2P networks like Gnutella. Flooding in a wireless network is in fact relatively efficient as compared to wired networks because of the wireless broadcast advantage, but there are also disadvantages which will be explained below.

Another possible approach for data dissemination is report pushing. Report pushing is the dual problem of report pulling; reports are flooded, and consumed by peers whose query is answered by received reports. So far there exist mechanisms to broadcast information in the complete network, or in a specific geographic area (geocast), apart from to any one specific mobile node (unicast/mobile ad-hoc routing) or any one arbitrary node (anycast). Report pushing paradigm can be further divided into stateful methods and stateless methods. Most stateful methods are topology-based, i.e. they impose a structure of links in the network, and maintain states of data dissemination. PSTree [42], which organizes the peers as a tree, is an example of topology based methods.
Another group of stateful methods is cluster- or hierarchy-based method, such as [43], in which moving peers are grouped into some clusters or hierarchies and the cluster heads are randomly selected. Reports are disseminated through the network in a cluster or hierarchy manner, which means that reports are first disseminated to every cluster head, and each cluster head then broadcasts the reports to the member peers in its group. Although cluster- or hierarchy-based methods can minimize the energy dissipation in moving peers, these methods will fail or cost more energy in highly mobile environments since they have to maintain a hierarchy structure and frequently reselect cluster heads.

Another stateful paradigm consists of Location-based methods (see [44]). In location-based methods, each moving peer knows the location of itself and its neighbors through some localization techniques, such as GPS or Atomic Multilateration (see [44]).

The simplest location-based data dissemination is Greedy Forwarding, in which each moving peer transmits a report to a neighbor that is closer to the destination than itself. However, Greedy Forwarding can fail in some cases, such as when a report is stuck in local minima, which means that the report stays in a mobile peer whose neighbors are all further from the destination. Therefore, some recovery strategies are proposed, such as GPSR (Greedy Perimeter Stateless Routing [45]). Other location-based methods, such as GAF (Geographic Adaptive Fidelity [46]) and GEAR (Geographical and Energy Aware Routing [47]), take advantage of knowledge about both location and energy to disseminate information and resources more efficiently.

In stateless methods, the most basic and simplest one is flooding-based method, such as [48]. In flooding-based methods, mobile peers simply propagate received reports to all neighboring mobile peers until the destination or maximum a hop is reached. Each report is propagated as soon as it is received. Flooding-based methods have many advantages, such as no state maintenance, no route discovery, and easy deployment. However, they inherently cannot overcome several problems, such as implosion, overlap, and resource blindness. Implosion refers to the waste of resources taking place when a node forwards a message to a neighbor although the latter may have already received it from another source. Overlap occurs when two nodes read the same product record or coupon, and thus push into the network
the same information. Resource blindness denotes the inability of the protocol to adapt the node’s behaviour to its current availability of resources, mainly power [49]. Therefore, other stateless methods are proposed, such as gossiping-based methods and negotiation-based methods.

Gossiping-based methods, such as [50], improve flooding by transmitting received reports to a subset of randomly selected neighbors; another option is to have some neighbours simply drop the report. For example, the neighbors that are not themselves interested in the report drop it. The advantages of gossiping-based methods include reducing the implosion and lowering the system overhead. However, dissemination, and thus performance, is reduced compared to pure flooding.

Negotiation-based methods solve the implosion and overlap problem by transmitting first the id's of reports; the reports themselves are transmitted only when requested (see [51]). Thus, some extra data transmission is involved, which costs more memory, bandwidth, and energy. In addition, in negotiation-based methods, moving peers have to generate meta-data or a signature for every report so that negotiation can be carried out, which will increase the system overhead and decrease the efficiency.

Another important stateless paradigm for data dissemination in mobile P2P databases is store-and-forward. In contrast to flooding, store-and-forward does not propagate reports as soon as they are received; rather they are stored and rebroadcast later. Store-and-forward methods include PeopleNet [37] and 7DS [38].

In summary, the paradigms for data dissemination in mobile P2P databases are summarized in Figure 4 below.
2.2 Experimental Projects

Table I: Pedestrians mobile peer-to-peer projects

<table>
<thead>
<tr>
<th>Pedestrians Projects</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>7DS</strong> – Columbia University ([38]) <a href="http://www.cs.unc.edu/~maria/7ds/">http://www.cs.unc.edu/~maria/7ds/</a></td>
<td>Focuses on accessing web pages in environments where only some peers have access to the fixed infrastructure.</td>
</tr>
<tr>
<td><strong>iClouds</strong> – Darmstadt University <a href="http://iclouds.tk.informatik.tu-darmstadt.de/">http://iclouds.tk.informatik.tu-darmstadt.de/</a></td>
<td>Focuses on the provision of incentives to brokers (intermediaries) to participate in the mobile P2P database.</td>
</tr>
<tr>
<td><strong>MoGATU</strong> – University of Maryland, Baltimore County <a href="http://mogatu.umbc.edu/">http://mogatu.umbc.edu/</a></td>
<td>Focuses on the processing of complex data management operations, such as joins, in a collaborative fashion.</td>
</tr>
</tbody>
</table>
**PeopleNet** – National University of Singapore ([37])

Proposes the concept of information Bazaars, each of which specializes in a particular type of information; reports and queries are propagated to the appropriate bazaar by the fixed infrastructure.

**MoB** – University of Wisconsin and Cambridge University
http://www.cs.wisc.edu/~suman/projects/agora/

Focuses on incentives and the sharing among peers of virtual information resources such as bandwidth.

**Mobi-Dik** – University of Illinois at Chicago
http://www.cs.uic.edu/~wolfson/htm/p2p.html

Focuses on information representing physical resources, and proposes stateless algorithms for query processing, with particular concerns for power, bandwidth, and memory constraints.

**Table II: Vehicular mobile peer-to-peer projects**

<table>
<thead>
<tr>
<th>Vehicular Projects</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CarTALK 2000</strong> – A European project</td>
<td>Develops a co-operative driver assistance system based upon inter-vehicle communication and mobile P2P databases via self-organizing vehicular ad-hoc networks.</td>
</tr>
<tr>
<td><a href="http://www.cartalk2000.net/">http://www.cartalk2000.net/</a></td>
<td></td>
</tr>
<tr>
<td><strong>FleetNet</strong> – Internet on the Road Project</td>
<td>Develops a wireless multi-hop ad hoc network for intervehicle communication to improve the driver's and passenger's safety and comfort. A data dissemination method called &quot;contention-based forwarding&quot; (CBF) is proposed in which the next hop in the forwarding process is selected through a distributed contention mechanism based on the current positions of neighbors.</td>
</tr>
<tr>
<td><a href="http://www.ccrle.nec.de/Projects/fleetnet.htm">http://www.ccrle.nec.de/Projects/fleetnet.htm</a></td>
<td></td>
</tr>
<tr>
<td><strong>VII</strong> – Vehicle Infrastructure Integration, a US DOT project</td>
<td>The objective of the VII project is to deploy advanced vehicle-to-vehicle (using the mobile P2P paradigm) and vehicle-to-infrastructure communications that could keep vehicles from leaving the road and enhance their safe movement through intersections.</td>
</tr>
<tr>
<td><a href="http://www.its.dot.gov/vii/">http://www.its.dot.gov/vii/</a></td>
<td></td>
</tr>
<tr>
<td><strong>Grassroots</strong> – Rutgers University ([39])</td>
<td>Develops an environment in which each vehicle contributes a small piece of traffic information to the network based on the P2P paradigm, and each vehicle aggregates pieces of the information into a useful picture of the local traffic information.</td>
</tr>
<tr>
<td><a href="http://paul.rutgers.edu/~gsamir/dataspace/grassroots.html">http://paul.rutgers.edu/~gsamir/dataspace/grassroots.html</a></td>
<td></td>
</tr>
</tbody>
</table>
In this project we introduce the general model considered throughout the dissertation and the principle of ranked store-and-forward.

3.1 General Model

A mobile P2P system is a set of point (i.e. without an extent) mobile peers \( M = \{M_1, M_2, \ldots, M_m\} \) capable of short range wireless communication. The capability is associated with a transmission range \( r \), which is the maximum physical distance between communicating peers. Peers that are within transmission-range are called neighbors.

Occasionally, a mobile peer \( M_i \) produces a report \( R \) having some unique report-id, and a size \( s(R) \). Reports are transmitted between neighbors. Each exchange of reports occurs within a single hop, and although there is no explicit multi-hop routing of reports, a report can propagate by multi-hop transmissions.

Each peer \( M_i \) has a (local) reports database \( RDB_i \), which stores the reports that \( M_i \) has produced or has received from neighbors (i.e., by P2P communication). The size limit of \( RDB_i \) is \( S \) bytes. When a report is produced or received by \( M_i \), if space is sufficient, the report is stored in \( RDB_i \). Otherwise, i.e. if space is insufficient, either the new report is not stored, or some reports are deleted from the database to accommodate the new report; the action taken depends on the storage management algorithm. We denote the global reports database \( RDB \), i.e., \( \bigcup_{i=1,2,\ldots,m} RDB_i = RDB \). Thus, each \( RDB_i \) is a subset of the reports in \( RDB \). At any point in time, the content of different \( RDB_i \)'s may overlap, i.e., it is possible that \( RDB_i \cap RDB_j \neq 0 \) for \( i \neq j \).

At any point in time \( t \), each peer \( M_i \) may have a query \( Q \) that represents the current interest of the peer. The query is continuous. For simplicity we ignore the case where multiple queries are asked, but this case can be easily handled. The query of \( M_i \) is called the query internal to \( M_i \), and \( M_i \) is called its originator.
query which is not internal is external. M_i's query is trivial if it is 'true', i.e. M_i requests all the reports in the RDB.

We assume that the degree of satisfaction between a report R and a query Q, denoted Q(R), is a value between 0 and 1. For example, if Q and R are two images, then Q(R) is the similarity between the two images. If Q(R)>0 we say that R satisfies Q. Given a report R and a peer M_i, if R satisfies M_i's internal query, then M_i is a consumer of R; if M_i produced the report then it is the producer of R. In any case, any peer is a broker of R in the sense that it can receive R from other peers and/or send it to other peers. Thus, M_i can be a producer, a consumer, and a broker of a report at the same time.

In addition to reports, each M_i also receives neighbors' queries. It accumulates these external queries in a queries (or demand) database QDB_i of N_i bytes. The demand database is FIFO maintained.

In terms of the communication model, we assume that at any point in time every peer knows its neighbors (i.e., the peers within its transmission range) by using a neighbor-discovery protocol. An encounter is the event in which a mobile peer M_i first detects a new neighbor. As long as the neighbor stays within transmission range M_i will not encounter it again, but it may do so after the neighbor disconnects.

3.2 Ranked Store-and-Forward

We advocate a store-and-forward mechanism, in which peers that serve as brokers of information save reports and queries, and propagate them upon encounters with other peers. Given the communication environment characterized above, we postulate that ranking of reports is important in mobile P2P databases, so that the most important reports are transmitted first. Therefore we propose a paradigm, ranked store-and-forward, as an approach to reports and queries dissemination in mobile P2P databases. Conceptually, the paradigm consists of neighbors (i.e., the mobile peers within the transmission range of each other) exchanging reports and queries. During an exchange, a mobile peer sorts the reports in its reports database according to their importance, and transmits the top k reports to its neighbors. The
frequency of exchanging and the size of each transmission (i.e. k) depend on the available bandwidth and power. Figure 5 gives an example of the ranked store-and-forward procedure.

**Figure 5.** An example of the ranked store-and-forward procedure. (a) Mobile peer A transmits its top two reports which are reports 1 and 4. (b) After receiving from A, peer B incorporates the received reports, re-ranks, and transmits the top two (shadowed). The same for C.

The ranked store-and-forward approach overcomes the barriers created by mobility and sparseness of peers. First, reports may be disseminated even if the network is disconnected. In other words, a mobile peer carries reports when it does not have any neighboring peers, and it forwards reports when a neighboring peer is encountered. Thus reports are epidemically disseminated from the producer to mobile peers that are out of the transmission range. Second, there is no need for a pre-configured data access or routing structure; data and queries are epidemically disseminated.

Ranked store-and-forward also overcomes the barriers created by bandwidth, energy, and memory constraints. Specifically, when energy and bandwidth are abundant, then ranked store-and-forward naturally reduces to flooding\(^2\) in the sense that each mobile peer caches all the reports that it receives, and it relays them to each other mobile peer that it encounters. However, observe that as new reports are generated, it becomes increasingly unlikely that a mobile peer has enough power and bandwidth to relay

\(^2\) Flooding is a network communication model in which a message received on each node is sent out to all the neighbors of that node except the one on which the message was received originally.
all the reports it has ever received. Then a prioritization mechanism, as provided by ranked store-and-forward, becomes critical.
In this chapter we discuss our results in the following aspects:

- MARKET query processing
- Application to parking slot discovery
- Application to blobs queries
- Application to KNN queries
- Pattern of reports propagation

### 4.1 MARKET Query Processing

The problem with store-and-forward algorithms is that the reports that need to be stored and forwarded by a peer may exceed its storage, bandwidth capacities. In this thesis we propose to address this problem by ranking of reports, so that the most relevant reports are transmitted and saved to the local database. The fundamental components for the rank-based store-and-forward are the following:

1. Supply, i.e., how many peers already have a report. The higher the supply of the report, the lower its relevance. We propose a machine learning algorithm called MALENA for the estimation of supply.

2. Demand, i.e., how many peers are querying the report. The higher the demand of a report, the higher its relevance.

In this thesis we integrate the above components into the MARKET algorithm that ranks reports based on supply and demand, and at each interaction communicates the “right” amount of information.

#### 4.1.1 Overview of MARKET

The MARKET algorithm is an integration of multiple mechanisms that enable each mobile peer to do achieve this objective. These mechanisms include:
1. **When to interact.** The query processing executed by MARKET consists of a sequence of send-and-receive interactions (see Figure 6). There are two types of interactions. The first type is *query-response (QR)*, which is triggered when a mobile peer encounters another mobile peer. The second type is *Relay*, which is triggered when transmission has not occurred for a pre-specified amount of time, and the mobile peer has new reports to disseminate. This *dual-type* mechanism makes MARKET automatically adapt to different mobility environments.

In a highly dynamic\(^3\) and/or partitionable environment, MARKET disseminates reports mainly via the encounters (QR interactions); in a static environment (where there are rare encounters), MARKET disseminates reports mainly via proactive transmission of newly produced reports (relay interactions).

2. **How to interact.** A QR interaction has two phases. In the first phase, the encountering mobile peers exchange their queries and receive answers. In the second phase, they transmit reports that enhance the other peer’s capability as a broker, i.e. reports that are in high demand but do not satisfy the received query. The reports are transmitted by broadcast so that the other neighboring peers may overhear the transmission, and thus their broker capability will also be enhanced. This reduces the communication cost when peer density increases. Thus, the QR interaction is a combination of one-to-one and broadcast communication, and the MARKET algorithm is a combination of report push and pull, in sense that the first phase of QR is pull, and "broker enhancement" and relay are push. The relay interaction of MARKET advertises the set of reports that were never transmitted, and if some of them satisfies the query of a neighbor, then the set is broadcast.

3. **What to transmit during an interaction.** Observe that since bandwidth is limited, not all the reports that satisfy the query or enhance the broker capability can always be transmitted. The number of reports that can be transmitted is determined by the underlying communication layer. Thus, ranking is used to determine which reports to transmit. The rank is also used by the receiving peer to accommodate the most popular reports in the limited space of the reports database. Intuitively, the rank of a report

---

\(^3\) Observe that there can be two reasons for an environment to be dynamic. One is high mobility. Another is high turn-over, namely the mobile peers frequently enter and exit the system.
depends on its size, demand (how many peers are querying it), and supply (how many peers already have it). For the estimation of demand, each mobile peer uses its demand database as a sample of the global demand. For the estimation of supply, we use the MALENA algorithm described in §4.1.2.4.

4. What to save. Given the limited space of the reports database, a mobile peer saves the reports that have the highest ranks. In other words, we assume that the answers received by the mobile peer are presented to the user, and possibly moved to the application area. Thus the reports saved in the reports database are solely for the purpose of brokering.

4.1.2 Reports Ranking by Supply and Demand

In §4.1.2.1 and §4.2.2.2 we define supply and demand, and discuss our proposed ranking method under the assumption that supply and demand are given. In the rest of §4.1.2 we discuss the estimation of supply and demand. In §4.1.2.3 we determine the minimum number of queries in the demand database
that is necessary to accurately estimate the demand. In §4.1.2.4 we discuss how to estimate the overall supply locally at a peer.

### 4.1.2.1 The Ranking Method

The rank of a report \(R\) at a peer \(M\) depends on the following three factors.

1. The demand of \(R\) at time \(t\), denoted \(\text{demand}(R,t)\), is the average degree to which \(R\) satisfies the query of a mobile peer in the system at time \(t\). In other words,

\[
\text{demand}(R,t) = \frac{\sum_{i=1}^{m} Q_i(R)}{m}
\]  

(3.0)

where \(m\) is the number of peers in the system. (Remember that \(Q(R)\) is the degree of satisfaction between a query \(Q\) and \(R\), see §3.1). The demand-database is used as a sample for the estimation of this demand. Formally, let \(O_1, O_2, \ldots, O_n\) be the queries in \(QDB\) (the demand database of \(M\)) at time \(t\). \(\text{demand}(R,t)\) is estimated by \(\text{demand}(R,t,M)\), defined as follows.

\[
\text{demand}(R,t,M) = \frac{\sum_{i=1}^{n} Q_i(R)}{n}
\]  

(3.1)

In §4.1.2.3 we establish the minimum size of this database such that Eq. 3.1 accurately estimates \(\text{demand}(R,t)\).

2. The supply of \(R\) at time \(t\), denoted \(\text{supply}(R,t)\), is the probability that an arbitrary peer has received \(R\) before time \(t\). This number is a global parameter that is normally unknown by each individual peer, but it can be evaluated by the peer based on metadata about \(R\) such as the number of times \(M\) received \(R\). The computation of the supply is described in §4.1.2.4.

3. The size of \(R\), denoted \(\text{size}(R)\). The smaller \(\text{size}(R)\), the higher the rank of \(R\); so to disseminate as many reports as possible. The rank of \(R\) at time \(t\) is

\[
\text{rank}(R,t) = \frac{\text{demand}(R,t) \cdot (1 - \text{supply}(R,t))}{\text{size}(R)}
\]  

(3.2)
The justification to the above ranking formula is given next.

**Ranking as an Approximation Method**

In this subsection we justify the rank formula (Eq. 3.2) by showing that it approximates an optimal solution to the NP-complete reports-selection problem.

Let $U$ be a set of reports stored at a mobile peer $M$. When selecting a subset of reports (to save or transmit) out of $U$, it is desirable that the selection adds as much throughput as possible to an arbitrary peer encountered in the future. Now we argue that $\text{demand}(R,t) \cdot (1 - \text{supply}(R,t))$ gives the degree of satisfaction that $R$ adds to an arbitrary peer $O$.

**Proposition 4.1.** Let $O$ be an arbitrary peer. If $R$ is received by $O$ at time $t$, the degree of satisfaction that $R$ adds to $O$ is a random variable with expected value $\text{demand}(R,t) \cdot (1 - \text{supply}(R,t))$.

**Proof:** $\text{demand}(R,t)$ is the degree to which $R$ satisfies the query of $O$. If $R$ has been received previously by $O$, then the degree of satisfaction added to $O$ is 0. The probability of $R$ having been received previously is $\text{supply}(R,t)$. The probability that $R$ has not been received by $O$ by time $t$ is $(1 - \text{supply}(R,t))$. Thus the expected degree of satisfaction that $R$ adds to $O$ is $\text{demand}(R,t) \cdot (1 - \text{supply}(R,t))$.

Let us refer to $\text{demand}(R,t) \cdot (1 - \text{supply}(R,t))$ as the **utility of** $R$. The **reports selection (RS) problem** is to construct a subset $U'$ of $U$, such that the sum of the utility values of the reports in $U'$ is maximized, subject to the constraint that the sum of the sizes of the reports in $U'$ does not exceed $T$. Intuitively, $U'$ is expected to add a higher amount of satisfaction to an arbitrary peer than any other subset of $U$ that does not exceed the size limit $T$. The Knapsack problem is straightforwardly transformed to the RS problem and thus the RS problem is NP-complete.

We use a well-known knapsack problem heuristic as [52] does: Sort the reports in the descending order of the ratio of utility to size (which is exactly the ranking function defined in Eq. 3.2). Then select the reports with the highest ranks, one by one, until no more can fit into the size limit $T$. We refer to this
algorithm as *Greedy RS* (or GRS). The time complexity of GRS is dominated by sorting $U$ and is $O(n \log n)$; $n$ is the cardinality of $U$.

Finally let us explain what Eq. 3.2 means for a newly produced report $R$. For such a report, MALENA assumes that its supply is zero. Thus its rank is $\frac{\text{demand}(R,t)}{\text{size}(R)}$.

### 4.1.2.2 Number of queries in the demand database

If we treat the demand database of a peer $M$ (i.e., $QDB$) as an arbitrary sample of the queries in the system, by [53], it can be shown that the deviation of Eq. 3.1 from $\text{demand}(R,t)$ is bounded as follows. For an arbitrary positive real number $\Delta$,

$$\Pr( | \text{demand}(R,t,M) - \text{demand}(R,t) | \leq \Delta ) > 1 - 2e^{-2n\Delta} \quad (3.3)$$

The equation says that the probability that [the difference between the $\text{demand}(R,t,M)$ and $\text{demand}(R,t)$ is smaller than $\Delta$] is greater than $1 - 2e^{-2n\Delta}$. The right-hand side of Eq. 3.3 is the confidence level. By setting it to the desired value, and setting $\Delta$ to the desired confidence interval width, we can solve for $n$.

For example, if the desired confidence level is 95% and the confidence interval width is $\Delta=0.08$, then $n$ should be set to 108. In this case the difference between $\text{demand}(R,t,M)$ and $\text{demand}(R,t)$ is smaller than 0.08 with probability 0.95. Suppose that the average query size is 100 bytes. Then the size of $QDB$ should be set to $108 \times 100 = 10K$ bytes.

### 4.1.2.3 Computing Supply by Machine Learning

In this subsection we outline an algorithm, called MALENA, for the computation of $\text{supply}(R)$. Due to space limitations we only give the main ideas of the algorithm.

To introduce the MALENA algorithm, observe that the supply of $R$ depends on attributes of $R$ (e.g. its age), as well as global system parameters such as the turnover rate (i.e. the rate at which peers enter and exit the system). The attributes of $R$ that can affect its supply are called *supply indicators*. It can be shown that, unfortunately, no single indicator is a good predictor of novelty in all environments. For example, in
some environments the intuition that the age of the report is a good predictor of supply is correct whereas
in some other environments (e.g. when the turnover is very high) it is not.

MALENA combines various novelty indicators in order to estimate the supply. The combination uses
machine learning to infer from previously received reports what the indicators of a new report “look like”. In
other words, it learns the supply based on the supply indicators of reports that it receives. The set of
novelty indicators of a report $R$, called the supply indicator vector (SIV) of $R$, is maintained by each
mobile peer $M_i$ that stores $R$. During an encounter, $M_i$ determines whether $R$ is new or old to the
encountered peer, and the respective SIV becomes a training example. In other words, $M_i$ treats the
encountered peer as an arbitrary mobile peer. In this fashion, a mobile peer progressively collects a
examples set which improves its learning system. When the peer ranks reports, the learning system is
used to calculate the supply. Furthermore, using a sliding window of examples MALENA adapts to new
environments.

The MARKET algorithm at a peer $A$ uses MALENA with the SIV consisting of two indicators for a
report $R$: 1) the age of $R$, and 2) the number of times the report has been encountered at a neighbour of $A$,
denoted $f_{in}$. The supply of $R$ increases as these indicators increase, thus the rank of $R$ decreases. The
machine-learning method used is Bayesian. Now we describe MALENA in detail.

**Description of MALENA**

As part of the MARKET-algorithm, before a mobile peer $O$ starts a reports transmission, it advertises
the id’s of the reports in $O$’s reports database. A neighbor peer $B$ replies with a subset of the advertised
id’s that includes the id’s of the reports that are new to $B$. The MALENA algorithm is invoked by $O$ when
it receives this reply from $B$. The main process of MALENA is to create examples and to train the
machine learning system. The pseudo-code of MALENA is given in Figure 7.
Algorithm: MALENA executed at O.

Input: $IDS_{ADV}$ the set of id’s advertised by O;
      $IDS_{REQ}$ the subset of id’s that are new (i.e. previously unknown) to the peer B.

Process:
For each report $R$ referred to by $IDS_{ADV}$ and its SIV $X$,
1. Create an example $(X, \text{label})$ where label is “new” if $R$’s id appears in $IDS_{REQ}$, and “old” otherwise.
2. Invoke INSERT_EXAMPLE((X, label)) to add the example (X, label) to the examples set.
3. If $R$’s id does not appear in $IDS_{REQ}$, update $X$ by increasing its $f_{in}$ value by 1.

Note: INSERT_EXAMPLE is where the machine learning system is actually trained. After the INSERT_EXAMPLE is finished, (X, label) is discarded. The INSERT_EXAMPLE procedure uses Bayesian learning.

Figure 7. Pseudo-code of MALENA algorithm executed at peer O (invoked in step 4 of QR explained in Figure 8)

Let us explain step 3 of the MALENA algorithm. If $R$’s id does not appear in $IDS_{REQ}$, then peer B must have already known $R$ before it encounters O. Thus $R$’s $f_{in}$ value is increased to reflect this fact.

Using Bayesian learning, the time complexity of the INSERT_EXAMPLE procedure is a constant. Thus the complexity of the MALENA algorithm is dominated by search in steps 1 and 3, and is linear in the cardinality of $IDS_{REQ}$ set.

Now we elaborate on the old/new labeling of the examples. Observe that a report may be received, then purged from the reports database, then received again. It would be false to label the report ‘new’ in the second receipt. But this is exactly what $M_i$ would do if the label is determined by simply considering the reports database. Thus, $M_i$ keeps a tracking set, in which each entry is the report-id of a report that has ever been received by $M_i$. An entry in the tracking set survives even when the corresponding report is purged from the reports database. And when a report is received, its report-id is searched in the tracking set for labeling, and thus “false” labeling is avoided. Furthermore, the tracking set enables $M_i$ to request
from the encountered peer only the reports that $M_i$ has never received, and thus avoid receiving old reports (pertaining to $M_i$).

Observe that the size of each entry in the tracking set is only a few bytes, thus the tracking set can contain many more entries than the reports database. Moreover, we have found that the cardinality of the tracking set can be bounded to $m \cdot G$ without significant performance degradation, where $m$ is the number of peers in the system and $G$ is the average number of members in the reports database of a mobile peer.
### Procedure: Query-response, executed at peer A when A encounters a peer B.

**Input:**
- $Q_A$ and $Q_B$ are the internal queries of A and B respectively.
- $IDS_A$ is the set of the id’s of the reports in $RDB_A$.
- $IDS_B$ is the set of the id’s of the reports in $RDB_B$.
- $TS_A$ and $TS_B$ are the tracking sets maintained by A and B respectively.
- $K$ is the maximum number of bytes that can be transmitted for the interaction.

**Process:**
1. Send $Q_A$ and $IDS_A$ to B by unicast.
2. Receive $Q_B$, $IDS_A$−$TS_B$, $IDS_B$−$IDS_A$ // By this step A knows what B wants ($Q_B$), what B does not know ($IDS_A$−$TS_B$), and what B has to offer ($IDS_B$−$IDS_A$).
3. Put $Q_B$ in the demand database $QDB_A$. // $QDB_A$ is FIFO-maintained.
4. Invoke MALENA to create examples and train the machine learning system. // The reports referred to by $IDS_A$−$TS_B$ are new to B, and those referred to by $IDS_A$∩$TS_B$ (i.e., $IDS_A$−($IDS_A$−$TS_B$) are old to B.
5. Fill up a message of $K$ bytes in the following order: // By this step A informs B what A wants ($Q_A$) and what A has to offer ($IDS_A$).
   - a) $IDS_B$−$TS_A$ // This is the set of the id’s of the reports in B’s reports database that are new to A.
   - b) The reports in $RDB_A$ that satisfy $Q_B$ and their id’s are in $IDS_A$−$TS_B$ (these are the answers to $Q_B$ that are unknown to B). If all the reports in this category do not fit in the message, they are selected in descending order of $Q_B$(R)/size(R). (GRS)
   - c) Other reports in $RDB_A$ whose ids are contained in $IDS_A$−$TS_B$ (these are the broker-enhancement reports). If all the reports in this category do not fit in the message, then the GRS algorithm described in §4.1.2.1 is executed to select the reports to include in the message
7. Symmetrically, A receives reports from B and puts them in $RDB_A$. If the size of $RDB_A$ is bigger than $S_A$ (Recall that $S_A$ is the size limit of $RDB_A$), then the GRS algorithm is executed to select the reports for saving.

**Figure 8.** Detail steps of Query-response interaction executed at peer A when A encounters a peer B

### 4.1.3 Experiments and Results

This section is organized as follows. In §4.1.3.1 we briefly describe the compared algorithms. In §4.1.3.2 we present the simulation environment and the performance measure. In §4.1.3.3 we present the simulation results.
4.1.3.1 The Compared Algorithms

We compare the performance of the MARKET algorithm with three other Mobile P2P caching schemes known as RANDI [11], LRU and LFU. All the compared algorithms adopt an identical sequence of send-and-receive interactions introduced by MARKET (i.e., QR and Relay interactions). However, they implement different ranking schemes. As mentioned previously, MARKET employs a ranking scheme to determine which reports to transmit during QR and Relay interactions. The rank is also used by the receiving peer to accommodate the most popular reports in the limited space of the report database. Hence, the compared algorithms differ from MARKET by incorporating different ranking paradigms while transmitting or accommodating the reports. Brief descriptions of the compared algorithms are as follows:

RANDI: RANDI ranks the reports solely based on their corresponding demand values. In other words, the difference between MARKET and RANDI is that MARKET ranks reports based on supply and demand, whereas RANDI ignores the supply.

LRU (least recently used): In LRU, the reports that satisfy the latest external query have the highest rank while ties are resolved based on the size (smaller size higher rank). The reports that do not satisfy the latest query are checked to see if they satisfy the second latest external query: those which satisfy it are ranked by their sizes, and those which do not are checked against the third latest external query. This procedure continues until all reports are ranked based on queries in QDB (clearly in the order of their arrival times).

LFU (least frequently used): LFU ranks the reports based on the query frequency. In other words, reports’ ranks in this scheme are proportional to the number of times that a report satisfies the arrived queries. Reports with the higher ranks are those which are accessed more frequently. Observe that for a newly produced report, LFU always ranks it very low because its query frequency is zero. MARKET and RANDI, on the other hand, are ready to estimate the demand of a newly produced report using the
previously collected query samples. If LFU were to save the arrived queries as well, then it would become the RANDI algorithm.

Finally, in order to put the performance of the algorithms in perspective, we define an *ideal benchmark* offline algorithm. In the ideal benchmark algorithm, when a report is produced, it is instantaneously disseminated to all the mobile peers currently in the system.

### 4.1.3.2 Simulation Environment

#### 4.1.3.2.1 Mobility Pattern

In order to simulate a sample real-world mobility pattern, we used a freely available archive of wireless network data recourses called CRAWDAD—Community Resource for Archiving Wireless Data At Dartmouth [54]. Amongst various wireless traces archived by CRAWDAD, we chose one of the latest datasets. It includes various real traces of Bluetooth sightings by 41 small devices (iMotes) carried by a conference attendees. Data in iMote dataset is *time-based* and not *location-based*. In other words, there is no information about the locations of different nodes at different times during the experiment. However, iMote dataset provides the set of the neighbors of each node at each unit of time (second). Accordingly, by looking over the iMote data, one can identify the *encountering* events at each unit of time, and consequently, determine whether MARKET QR or Relay interactions should be triggered. In all the simulations we used the first 50 minute part of the iMotes data set.

To our knowledge, all publicly available network-simulators assume a two-dimensional environment and associate an \((x, y)\) coordinate to each node. Since such assumption is not compatible with the time-based format of the iMote dataset, we have implemented an iMote-based simulator from scratch.

#### 4.1.3.2.2 Generation of Reports and Queries

For representing reports and queries, we adopted the Number Intervals (NI) subscription model introduced in [42]. Particularly, a report is represented by a point within the real interval \([0, 1]\). A query is represented by a range within \([0, 1]\), e.g., \([0.2, 0.7]\). A report \(R\) matches a query \(Q\) if \(R\)’s number falls into
$Q$’s range. Reports are produced by a Poisson process with intensity $P$. Each report’s value is randomly chosen from the [0, 1] interval. A number of arbitrary mobile peers become the producers of the report $R$. This simulates, for example, the scenario in which an event is monitored simultaneously by multiple mobile sensors.

### Table III. Simulation parameters and their values

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation time (secs)</td>
<td>$3000 = 50$ (min)</td>
</tr>
<tr>
<td>Number of peers</td>
<td>40</td>
</tr>
<tr>
<td>Relay time-interval (secs)</td>
<td>60</td>
</tr>
<tr>
<td>RDB max-size (KByte)</td>
<td>200</td>
</tr>
<tr>
<td>Report size range (KByte)</td>
<td>[2, 20]</td>
</tr>
<tr>
<td>QDB max-size (KByte)</td>
<td>1</td>
</tr>
<tr>
<td>Query size (Byte)</td>
<td>100</td>
</tr>
<tr>
<td>LFU sliding window size (secs)</td>
<td>1000</td>
</tr>
<tr>
<td>Transmission size4 (KByte)</td>
<td>[20, 200]</td>
</tr>
<tr>
<td>Report production rate (reports/sec)</td>
<td>$1/60 = .0166$</td>
</tr>
<tr>
<td>Query production rate (reports/sec)</td>
<td>$1/300 = 0.0033$</td>
</tr>
<tr>
<td>Mean and Std. of query length ($\mu, \sigma$)</td>
<td>(0.05, 0.002)</td>
</tr>
<tr>
<td>Number of producers for each report</td>
<td>3</td>
</tr>
</tbody>
</table>

Each mobile peer dynamically produces internal queries following a Poisson process with a predetermined intensity. Once a new query is generated, the old one will be expired. Thus every mobile peer has a single query at each unit of time. The range of the query is generated by choosing a center and a length. The length of the range is selected randomly according to a normal distribution with mean $\mu$ and $

$^4$The transmission size (referred by $K$ in the text) corresponds to the max amount of information which can be transmitted on each interaction.
standard deviation \(\sigma\). The query-center falls into the \([0, 1]\) interval following a Zipf distribution. In particular, the \([0, 1]\) interval is divided into 10 disjoint sections \(([0, 0.1), [0.1, 0.2), \ldots]\). The probability that a query-center falls into the \(i^{th}\) \((1 \leq i \leq 10)\) section is \((1/i)/\sum_{i=1,2,\ldots,10}(1/i)\). In other words, the resources are uniformly distributed, and the queries are distributed based on Zipf's law.

Table III lists the simulation parameters and their respective assigned values.

### 4.1.3.2.3 Performance Measure

The performance of different caching schemes has been evaluated in terms of the average number of distinct matches received by a mobile peer with response-times smaller than a certain time limit \(w\). The response-time of a report \(R\) received at a mobile peer \(O\) with query \(Q\) is defined as follows. The response time starts at the time at which either \(Q\) is generated by \(O\) or \(R\) is fed into the system, whichever is later (since both must be present), and ending when \(O\) receives \(R\). This measure is called the response-time bounded throughput, or throughput, and \(w\) is called the response-time bound. By varying the value of \(w\), we evaluate the throughput of an algorithm under different response-time constraints.

### 4.1.3.3 Simulation Results

In §4.1.3.3.1 we compare the performance of different caching schemes in terms of the response-time bounded throughput. In §4.1.3.3.2 and §4.1.3.3.3 we analyze the impacts of the transmission size and peers’ memory size (RDB size), respectively. Due to the stochastic nature of reports and queries, all the experiments have been repeated 20 times, and then their achieved performances were averaged. Hence, all the results presented in upcoming sections are the averages of 20 independent runs. Note that the defaults values of the simulation parameters for all the experiments are set up in Table III. Therefore, for investigating the impact of a single parameter (e.g., transmission size or RDB size), all other parameters retain their default values accordingly.

Estimation of demand and supply impose an additional space overhead which should be considered in our simulation in order to have a fair comparison. MARKET requires an extra space to 1) construct
Conditional Probability Tables (CPTs) of the employed Bayesian machine learning algorithm, 2) accommodate queries in peers’ query databases, and 3) maintain tracking sets (TSs) of report ids.

In order to consider MARKET’s space overhead, we subtract the average amount of required extra space from the available memory (RDB) of each mobile peer. Experimentally we have observed that the overall space overhead of the MARKET algorithm does not exceed 3KB.

4.1.3.3.1 Comparison for Given Parameter Setting

![Figure 9. Response-time bounded throughput of compared algorithms (LFU and RANDI overlap each other). Transmission size = 150 KB, RDB size=200KB](image)

Figure 9 shows the response-time bounded throughput of the compared algorithms. As mentioned previously, the response-time bound (x-axis) only extends to 50 (minutes) because all the curves are flat after this bound. Thus the throughput with response-time bound 50 represents the throughput regardless of the response-time-bound.
The MARKET throughput corresponding to the response-time bound of 10 minutes is 75% of the ideal performance. This means that, on average, 75% of the reports which are accessible immediately in an ideal situation are also accessible within 10 minutes in an environment where MARKET caching scheme has been employed. 25% of the reports are missed in MARKET due to insufficient peer density and memory/bandwidth constraints.

4.1.3.3.2 Impact of local memory (RDB) size

Figure 10 shows the overall averaged throughput (throughput with response-time bound of 50 minutes) of different methods while varying the RDB size of mobile peers. As the throughput graphs demonstrate, the different methods perform closely when RDB size is very small (50 KB). When peers have a very limited local memory, most of the reports will be purged; thus ranking scheme does not have enough opportunity to reveal its power over the other compared methods. When the RDB size is between 100KB and 200KB, MARKET outperforms all the other three mobile P2P schemes. When the RDB size is bigger than 250KB, LRU outperforms MARKET although MARKET still outperforms LFU and RANDI. This suggests that MARKET is particularly suitable for the situation where the database size is small.

Observe that the performance of MARKET changes more radically when the RDB size is between 50 KB and 200 KB. These are the settings where RDB size is fairly limited, and accordingly, ranking schemes affect algorithms’ performance.
Figure 10. Throughput as a function of RDB size. Transmission size = 100 KB.

4.1.3.3 Impact of transmission size

Figure 11 depicts the average throughput of the compared algorithms in terms of the transmission size (K). From the figure it can be seen that as the transmission size increases, the throughput increases for MARKET, RANDI, and LFU whereas the throughput of LRU remains almost constant. This suggests that MARKET is particularly suitable for the situation where the transmission size is big.
4.1.4 Relevant Work

**Static-sensor databases:** A database approach has been applied to static sensor networks (e.g., [55]). These methods require that a certain routing structure, e.g. a tree, be established in the network. Such a method is unsuitable for a dynamic network topology because the routing structure is hard to maintain. Some works in static sensor networks involve negotiations between neighboring sensors (e.g., [56]). The objective is to ensure that the reports wanted (i.e., unknown reports in our context) by the receiver are transmitted. However, the negotiation considers only the utility of reports to the receiver, whereas the MARKET-algorithm considers both the utility of reports to the receiver and to the future encountered mobile peers.

**Prioritization in mobile peer-to-peer data dissemination:** Ranking reports for memory (cache) management and bandwidth management in mobile peer-to-peer networks has been studied in a number of works. In [57] the rank of a report for storage only is jointly determined by its demand, reliability, and size, but not on supply. Our comparison with RANDI shows the importance of supply. In [58] reports are ranked based on an abstract utility function which is to be defined by specific applications. Our ranking method can be considered as an instantiation of the utility function.

**Delay/Fault-Tolerant Mobile Sensor Networks [59]:** This work studies how to efficiently deliver reports from sensors to sinks in disconnected mobile sensor networks. It is assumed that every sink is interested in receiving every sensor-produced report. In our context, there are queries and they may be different for different sinks, and these have significant implications in the P2P interaction mode and reports ranking.

**Resource discovery (e.g. [41]) and Publish/subscribe (e.g. [42]) in mobile P2P networks:** These papers often build a routing structure for resource information dissemination. Consequently they can be inefficient, particularly in networks that are prone to frequent topology changes and disconnections due to mobility and turn-over. In such an environment, either a lot of communication has to be expended to keep the routing structure up to date, or the routing structure rapidly becomes obsolete and misses many
matches. Furthermore, these methods depend on network connectivity, and do not work in sparse networks.

**Cooperative caching in mobile environments:** The MARKET algorithm performs a form of cooperative caching; the local database of each mobile peer is a cache that services a query originator in the QR operation. Most of existing cooperative caching methods are based on query-to-data (see e.g., [60]). If there does not exist a path between the query originator and the data caches, the query fails. This strategy suffers in a sparse environment, which is dealt with here by store-and-forward.

### 4.2 Application to Parking Slot Discovery

#### 4.2.1 Application Environment

In our environment, there is a single type of spatio-temporal resources, such as parking slots, car accidents (reports about such resources provide traffic-jam information), taxi-cab requests, ride-sharing invitations, or demands of expertise in disaster situations, and so on. These resources are spatial in the sense that they are tied to a location, and are temporal in the sense that they are valid or available only for a limited time-duration. We assume that resources are located at points in two-dimensional geospace. The state of each resource alternates between valid (i.e. available) and invalid. The period of time during which the resource is valid is called the valid duration. For example, the valid duration of the cab request resource is the time period since the request is issued, until the request is satisfied or canceled.

The validity of a resource $R$ is indicated by its validity report (or report for short), denoted $a(R)$. The report may be produced by a sensor or a processor associated with the resource. Each report $a(R)$ contains three attributes, namely resource-id, timestamp, and location. Attribute resource-id is the identification of $R$ that is unique among all the resources in the system. Timestamp indicates the time at which $a(R)$ is transmitted by $R$. Location indicates the location of $R$.

In addition to resources, the system consists of moving objects. At each point in time, a moving object $o$ is either a consumer or a broker. $o$ is a consumer if it is searching for a resource. $o$ is a broker if it is not
searching for a resource but is participating in MARKET dissemination. Resources are used only by moving objects that are consumers. \( o \) has a *reports database* that stores the reports \( o \) has received. Moving objects and resources are collectively called *peers*.

Each report \( a(R) \) has a relevance when it is received. The relevance of \( a(R) \) to a moving object \( o \) is determined by the following spatio-temporal function.

**Definition:** The relevance of a report \( a(R) \) to a consumer that receives it \( t \) time units after \( a(R) \)'s timestamp, and \( d \) distance units from the location of \( R \) is:

\[
\text{Rel}(a(R)) = e^{-\alpha \cdot t - \beta \cdot d} \quad (\alpha, \beta \geq 0) \quad (4.2.1)
\]

\( \alpha \) and \( \beta \) represent the decay factors of time and distance respectively. \( t \) represents the delay from the time when \( a(R) \) is transmitted until \( a(R) \) is received by the consumer, and is referred to as the *report delay*. We now show that for a competitive resource \( R \), under some very reasonable conditions the relevance of a report \( a(R) \) to a consumer \( o \), as computed by Equation (4.2.1), equals to the probability that \( R \) is still valid when \( o \) reaches it.

**Theorem 4.1:** Assume that consumers arrive at a resource \( R \) according to a Poisson process with intensity \( \lambda \). Let \( o \) be a consumer that moves at a constant speed \( v \), and receives a report \( a(R) \) \( t \) time units after \( a(R) \)'s timestamp, at distance \( d \) from the location of \( R \). When \( \alpha = \lambda \) and \( \beta = \lambda / v \), \( \text{Rel}(a(R)) \) as computed by (5.1) equals to the probability that \( R \) remains valid when \( o \) reaches it.

**Proof Idea.** Let \( t_0 \) be the timestamp of \( a(R) \). According to the report transmission model, \( R \) is valid at \( t_0 \). Since consumers arrive at \( R \) according to a Poisson process with intensity \( \lambda \), the probability that no other consumers reach \( R \) \( x \) time units after \( t_0 \) is \( e^{-\lambda \cdot x} \). Observe that the consumer \( o \) will reach the resource \( t+d/v \) time units after \( t_0 \), and thus the probability that \( R \) remains valid when \( o \) reaches it is \( e^{-\lambda \cdot (t+d/v)} \), which equals to \( \text{Rel}(a(R)) \).

The theorem motivates our definition of the relevance function.

The relevance function we use in this thesis is one example in which the relevance decays exponentially per time and distance. But there are other possible types of relevance functions in which
other behaviors may be exhibited. Furthermore, other factors such as the travel direction with respect to the home of a resource, or the price of the resource, may be considered in the relevance function. However, in this thesis we confine ourselves to time and distance alone.

4.2.2 Resource-discovery Strategies

In this subsection, we discuss two resource-discovery strategies for competitive resources; one does not use any reports, while the other one takes advantage of the reports of competitive resources.

**Blind Search.** The first competitive resource-discovery strategy is a naive one, called *blind search*, or BS. With this strategy, a consumer moves around the area where a resource of interest could possibly be located, and it takes possession of the first resource that is valid at the time when the consumer reaches it. For example, a driver who is looking for a parking slot simply drives around on the streets that are within walking distance from the destination, and parks at the first parking slot that is available when passed by. The area within which the consumer looks for a resource is referred to as the *search space*.

**Information Guided Search.** The second strategy is *information guided search*, or IGS. With this strategy, a consumer starts with a blind search. The search continues until either a valid resource is encountered (i.e. passed by in the road network), or some resource-report $a(R)$ is received. In the latter case, the consumer attempts to capture $R$ (i.e. moves along the shortest path to $R$). If $R$ is invalid when the consumer reaches it, then the consumer discards $a(R)$, returns to the closest point in the search space, and continues the blind search. Clearly, if a valid resource is passed by on the way to $R$, then the consumer captures it and the search ends. If another report $a(R')$ is received during the trip to $R$, and the relevance of $a(R')$ is higher than $\text{Rel}(a(R))$, then the consumer goes to $R'$. Thus, the relevance function plays an important role in the use of resource-discovery information.
4.2.3 Value of Resource Information

In this section, we evaluate how much time is saved when a consumer uses resource-reports to capture a competitive resource (resource-reports are disseminated by the MARKET dissemination mechanism). First we describe the simulation method. Then we present the simulation results.

4.2.3.1 Simulation Method

Evaluation Metrics. We use the discovery time as the metrics for evaluating the benefit of the resource-discovery strategies. For a competitive resource, discovery means that the consumer captures the resource, i.e. it arrives at the resource while the resource is still valid. For example, discovering a parking slot means that the driver reaches the parking slot before it is occupied. The discovery time is the length of the time period starting when the consumer starts to search the resource type and ending when a resource of that type is captured. Traditionally, the effectiveness of a data dissemination algorithm is measured in terms of its throughput (how many resources are found) and the response time (i.e. the time it takes on average to find a resource). The resources addressed in this thesis, whose state alternates between valid and invalid, enable us to combine the two measures into a higher level one, namely the discovery time.

Simulation Environment. We implemented our own simulation system in Java. First we describe the simulation of mobility and resources, and then we describe the simulation of wireless communication.

Simulation of Mobility and Resources. We synthetically generated and moved objects within a 1.2mile×1.2mile grid network. The distance between two neighboring grid points is 0.1 mile (approximately the length of one street block) (see Figure 12). Resources are generated at all non-border four-way intersections.
Each consumer $o$ is introduced at a random location on the grid network. $o$ is assigned a square as its search space, such that (i) the side length of the square is 0.6 mile; (ii) $o$ is initially located on one of the four edges (i.e. north edge, east edge, south edge and west edge) of the square with equal probability; (iii) the square is aligned with the grid network such that $o$ is as close to the middle of the edge as possible. The square is referred to as the search square of $o$. $o$ moves along its search square to search resources either clockwise or counter-clockwise with equal probability. With IGS, consumers may leave the search square to capture resources that are inside or outside the square. $o$ moves at a constant speed. The motion speed of $o$ is randomly picked up from the interval $[v-5, v+5]$ where $v$ is a parameter. Initially, $c$ consumers are introduced. Out of these consumers, fraction $k$ use the IGS strategy (referred to as IGS consumers), and the others use the BS strategy (referred to as BS consumers). $k$ is referred to as the IGS consumer ratio. Whenever an IGS (or BS) consumer captures a resource, and is thus eliminated from the system, a new IGS (or BS) consumer is introduced. Consequently, at any point in time there is a fixed number, $c$, of consumers in the system, fraction $k$ using resource information.
There are \( g \) brokers per square mile. The mobility of each broker \( B \) is simulated as follows. We randomly choose two points on the grid network, and assign them as the start point and the first stop of \( B \) respectively. The path of \( B \) is the shortest path between the start point and the first stop. \( B \) moves along its path from the start point to the first stop at a constant speed. When the first stop is reached, another random point is chosen as the second stop of \( B \), and \( B \) moves from the first stop to the second stop at the same constant speed. And so on. The motion speed of a broker is randomly chosen from the interval \([v-5, v+5]\).

After a resource \( R \) is captured and thus becomes invalid, there is a time period until another resource is generated at the same intersection as \( R \). This time period is referred as an invalid duration. The length of invalid duration follows an exponential distribution with mean \( q \).

We use equation (4.2.1) with \( \alpha = \lambda \) and \( \beta = \lambda / v' \) as the relevance function, where \( v' \) is the motion speed of the consumer, and \( \lambda = 2\cdot c\cdot v/l \); \( l = 31.2 \) miles is the total length of all the edges in the grid network, \( c \) is the number of consumers, and \( v \) is the average speed of consumers. Observe that in computing \( \beta \) we use the actual speed of the consumer (namely \( v' \)), which is randomly distributed around the mean \( v \), whereas in computing \( \lambda \) we use the mean \( v \). When computing the relevance, we use the route-distance as the distance metric. The route-distance between two locations on the grid network is the length in miles of the shortest path between them on the grid network. We argue in the appendix that with our simulation setup the arrival of consumers at a resource approximates a Poisson process with intensity \( \lambda \). Furthermore, we traced in simulations the arrival of consumers at each resource and found that the arrival process indeed approximates a Poisson process with intensity \( \lambda \). Thus, according to Theorem 4.1 the relevance function gives the probability that the resource is valid when the consumer reaches it.

*Simulation of Communication.* We assume that each moving object allocates only fraction \( a \) of the available short-range bandwidth to the simulated resource type. \( a \) is a parameter and is referred to as the bandwidth allocation. When computing the broadcast period, we use the data of 802.11b, and therefore the length of a time slot is 20\( \mu \)s. The data transmission speed is chosen to be 5.5Mbps. The transmission
range is 150 meters. The size of each resource report is 32 bytes. The node density varies from 150 to 450 objects/mile$^2$, depending on the broker density $g$ and the number of consumers $c$. The broadcast size varies from 1 to 121. All the parameters for the simulation system are illustrated in Table IV.

Our simulation system omits detailed representation of protocol layers and radio propagation. It models the reliability of communication by each neighbor correctly receiving a broadcast with certain probability. We consider two factors, i.e. (i) signal collisions due to hidden nodes; (ii) deteriorative channel conditions due to relative motion. To model the communication failures caused by signal collisions, we adopt the analytical model proposed in [61]. The analytical model computes the probability that a broadcast message is received by all of the sender’s neighbors without suffering any collisions. Such a broadcast is referred to as a successful broadcast. In our simulation system, if a broadcast is not successful, then none of the sender’s neighbors receives the broadcast message.

A successful broadcast is correctly received by each neighbor with certain probability depending on the relative speed of the sender and the receiver. The probability is referred to as the reception probability. We adopt the empirical results of [62] to determine the reception probability. Specifically, given the relative speed $s$, the ratio between the 802.11b throughput under $s$ and that under relative speed 0 is obtained from [62]. This ratio is taken to be the reception probability.
### Table IV. All parameters and their values

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Unit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean of invalid duration</td>
<td>$q$</td>
<td>minute</td>
<td>10, 15, 20, 25, 30</td>
</tr>
<tr>
<td>Broadcast size</td>
<td>$M$</td>
<td>report</td>
<td>1, 5, 10, 30, 50, 70, 90, 110, 121</td>
</tr>
<tr>
<td>Transmission range</td>
<td>$r$</td>
<td>meter</td>
<td>150</td>
</tr>
<tr>
<td>Motion speed</td>
<td>$v$</td>
<td>miles/hour</td>
<td>10, 20, 30, 40, 50</td>
</tr>
<tr>
<td>Number of consumers</td>
<td>$c$</td>
<td>objects/mile$^2$</td>
<td>50, 100, 150, 200, 250</td>
</tr>
<tr>
<td>IGS consumer ratio</td>
<td>$k$</td>
<td></td>
<td>0 to 1 with increment 0.1</td>
</tr>
<tr>
<td>Broker density</td>
<td>$g$</td>
<td>objects/mile$^2$</td>
<td>0, 50, 100, 150, 200</td>
</tr>
<tr>
<td>Data transmission speed</td>
<td>$b$</td>
<td>bits/second</td>
<td>$2 \times 10^6$</td>
</tr>
<tr>
<td>Medium access time slot</td>
<td>$\tau$</td>
<td>second</td>
<td>$20 \times 10^6$</td>
</tr>
<tr>
<td>Medium access control header</td>
<td>$h$</td>
<td>byte</td>
<td>47</td>
</tr>
<tr>
<td>Report size</td>
<td>$S$</td>
<td>byte</td>
<td>32</td>
</tr>
<tr>
<td>Bandwidth allocation</td>
<td>$a$</td>
<td></td>
<td>0.001, 0.002, 0.01, 0.1, 1</td>
</tr>
</tbody>
</table>

**In summary, each simulation run is executed as follows.** At the beginning of the simulation run, 121 resources are generated, each at one four-way intersection. $c$ consumers and $g \times 1.2 \times 1.2$ brokers are introduced at time 0 (remember that the total area simulated is 1.44 square miles) at random locations. Fraction $k$ of the consumers are IGS consumers and the others are BS consumers. A resource or broker $o$ periodically ranks the reports in its local reports database and broadcasts the top $M$ reports to its neighbors. $M$ is determined using the transmission size formula introduced in [21]. Each broadcast is correctly received by a neighbor with certain probability as described above. The sending and receiving of each broadcast is completed instantaneously (i.e. they take 0 time). The simulation run terminates after twenty simulated hours, out of which the first 500 seconds is the warm-up time period for the system to stabilize.

During each simulation run, the discovery time of each consumer is collected. The resource-discovery times of all the IGS consumers and the discovery times of all the BS consumers are averaged respectively.
In the conducted simulation runs, the ratio between the 95% confidence interval and the simulation result (i.e. the average discovery time) ranges from 4% to 19%, with the average ratio being 6.6%.

4.2.3.2 Simulation Results

Impact of the Broadcast Size on IGS (Figure 13). Figure 13 shows that increasing the broadcast size does not improve the performance of IGS. In other words, with the MARKET algorithm, broadcasting only the top one report is as good as broadcasting the whole database (121 reports). This is because MARKET chooses the broadcasted reports based on their spatio-temporal relevance which reflects the benefit of the reports. The fact that broadcasting the top one report is enough is a nice property. It indicates that MARKET is efficient in bandwidth consumption, and that it is drastically different than flooding that would broadcast all the reports in the database.

Overall comparison between IGS and BS. Figures 13 to 19 show the performance of IGS and BS under different parameter setups. There are three curves in each figure. Two of them represent the discovery times of IGS and BS collected in the same simulation run. The third curve, referred to as benchmark BS, represents the discovery time of BS under the same parameter setup, except that the consumer ratio \( k = 0 \). Thus benchmark BS represents the performance of BS when there are no IGS consumers in the system. From Figures 13 to 19 we can see the following. (i) IGS consistently outperforms both BS and benchmark BS. Sometimes IGS reduces discovery time by 70% compared to BS. (ii) Benchmark BS is always better than BS. This is because the IGS consumers capture resources faster than the BS consumers in the same system, which makes the BS consumers spend more time on searching than if there are no IGS consumers.
Figure 13. Discovery time versus broadcast size

Figure 14. Discovery time versus IGS consumer ratio
Figure 15. Discovery time versus number of consumers

Figure 16. Discovery time versus mean of invalid duration
Figure 17. Discovery time versus mean of motion speed

Figure 18. Discovery time versus broker density
IGS consumer ratio (Figure 14). The discovery time of IGS increases as the IGS consumer ratio increases. This is because as the IGS consumer ratio increases, more consumers use reports. Thus for each individual IGS consumer, the chance to capture a reported resource decreases. In other words, in competitive situations the value of resource information decreases as more consumers have access to this information. Further observe that the discovery time of BS also increases as the IGS consumer ratio increase. This is because the more IGS consumers, the more likely it is that a resource is captured by an IGS consumer, and thus the less chance for a BS consumer to capture a resource.

Number of consumers (Figure 15). The discovery time of IGS and that of BS both increase as the number of consumers increases. However, the curve of IGS is flatter than that of BS. In other words, IGS is less sensitive to the increase of competition introduced by the increase of the number of consumers. This suggests that resource information is especially valuable in the areas where consumers are dense (such as downtown), i.e. competition for resources is fierce.

Mean of invalid duration (Figure 16). The discovery time of IGS increases as the mean of invalid duration increases. So does the discovery time of BS. However, the discovery time of BS increases faster.
than that of IGS. In other words, IGS is less sensitive to the increase of competition introduced by scarcity of resources. Thus, again, the value of information increases as competition for resources increases.

**Mean of motion speed (Figure 17).** Observe that an increased mean motion speed has a negative impact on IGS. This is somehow surprising because a higher speed is supposed to enable a consumer to capture a resource faster. However, higher mobility also leads to more deteriorative channel conditions and thus lower broadcast reception probability. Thus IGS consumers are less advantageous against BS consumers.

**Broker density (Figure 18).** The discovery time of IGS decreases as the broker density increases. Intuitively, the increase of the broker density generates two contrary effects on performance. On the one hand, the reliability of broadcast decreases due to higher contention and more deteriorative channel conditions. On the other hand, each successful broadcast is likely to reach more objects. Figure 18 shows that the positive effect outweighs the negative one. Thus as the broker density increase, the newly generated reports are propagated more quickly and reach the consumer sooner, giving the consumer a higher probability of capturing a resource. In other words, the faster information spreads, the higher its value.

**Bandwidth allocation (Figure 19).** Figure 19 plots the performance of IGS as a function of the bandwidth allocation for a particular parameter configuration. The discovery time of IGS is higher with partial capacity than that with the full capacity, but still lower than that of BS. Particularly, the discovery time of IGS increases from 7.5 minutes to 9 minutes as the bandwidth allocation (fraction) decreases from 1 (corresponding to 5.5Mbps baseline bandwidth) to 0.002 (corresponding to 11Kbps baseline bandwidth), and to 11 minutes as the fraction decreases to 0.001 (corresponding to 5.5Kbps baseline bandwidth. The reason for the increased IGS discovery time is that with the reduced bandwidth the broadcast period increases (to 5 seconds with 11Kbps and 10 seconds with 5.5Kbps). Further observe that the performance of IGS degrades very slowly as the bandwidth allocation decreases. This suggests that the bandwidth
consumption of the MARKET algorithm for one resource type is far below the full network capacity and therefore it is able to support many resource types and other network applications.

4.2.4 Relevant Work

A lot of works have been done in data dissemination in mobile P2P networks. Most of these works are concentrated on how to disseminate resource information, whereas we study how much benefit the resource information may generate. In technical report [63], Goel et al. propose an architecture for dissemination of traffic information in mobile P2P environments and evaluate the benefit of traffic information in terms of the reduction of travel time. Their approach is geared to traffic information. For example, vehicles generate traffic reports only when the expected travel time on the road segment differs significantly from the travel time actually experienced by the vehicle, whereas we consider general spatio-temporal resources. Furthermore, they do not study how to use the resource information, whereas we do so in this thesis.

4.3 Application to Blobs Queries

We consider a Mobile P2P Network of smart-phones, pdas, or vehicles called peers (also known as nodes). Each peer has both WiFi and cellular communication capabilities, and it also has a database of binary large objects (blobs) such as songs, video clips, and/or computer programs. The blobs may be generated dynamically, e.g. a 2-3 seconds video of the surroundings is generated by a peer every minute. Additionally, the peers generate queries requesting blobs in the Mobile P2P Network. As in the network of shoppers in a mall, or passengers in an airport, or vehicles on the highway, a peer does not initially know the network-id’s (i.e. cell-phone numbers) of the other peers in the network. However, a peer can communicate directly with other peers within its WiFi\textsuperscript{5} transmission range without knowing their network-id.

\textsuperscript{5} We use the term WiFi for simplicity, but most results of this paper apply to other short-range networking technologies such as Bluetooth and DSRC.
In this thesis we propose a query processing paradigm that disseminates blobs and queries by mobile peer-to-peer (P2P) communication. Mobile P2P communication can be either by WiFi (short range), or by cellular infrastructure (long range), or by a combination of the two. The WiFi communication uses standard mobile ad-hoc dissemination protocols and techniques. The paradigm does not require or assume a central server. As in Vehicular Networks, we assume that energy is not a problem.

Considering the potential volume of blob data and the unpredictability of mobile P2P networks (e.g., partitions, high mobility), efficient querying is very challenging. We abstract the problem as follows. A database consisting of *blob reports* is geographically distributed among a set of *mobile peers*. Each report consists of a *blob* sub-report, and a *metadata* sub-report which includes the description of the blob (e.g., time and location at which a multimedia clip was produced). Queries are disseminated as *query-reports*. The match (yes or no) between a query and a report is determined solely based on the *metadata* sub-report.

In this thesis we conduct an analysis of blob query-processing strategies in mobile P2P networks. The reason for the multitude of strategies is that choices need to be made along the following design dimensions. First, the mobile P2P communication may use purely WiFi or it may use both WiFi and cellular communication (i.e., hybrid). Pure cellular communication is not an option because it requires knowledge of network id of the receiver, whereas a query originator initially only knows the description of the requested reports but not the network-id of the peers containing those reports.

Second, the query processing may adopt push (data-to-query) or pull (query-to-data). In the push fashion, blob reports are proactively disseminated. In the pull fashion, queries are proactively disseminated; blob reports are disseminated as responses to received queries. Push and pull can also be combined. Third, due to size-differences, the metadata and blob sub-reports of a given report may be disseminated independently, and by different means.

In this thesis we define a paradigm called (WiFi-communication, Match, Communication) or **WiMaC**, of query processing strategies based on the above design dimensions. Then we derive 13 different WiMaC query processing strategies, and in order to compare them we define the notion of dominance of one strategy by another. Then we prove analytically that four strategies dominate the
others. Finally, we compare these four strategies by simulation in a vehicular environment. It turns out that one is superior in terms of throughput. It is the one which disseminates queries and metadata sub-reports by WiFi; when a matching metadata and query meet at a peer, the corresponding blob report is transferred to the query originator by cellular communication.

4.3.1 Application Environment

4.3.1.1 Mobile Peers

The environment is a system consisting of a set of mobile peers, or peers. This set of peers may change over time. Each peer (e.g. a cell phone) is equipped with the following capabilities: (i) producing a blob (binary large object) data such as video, voice, or multimedia clips; (ii) short-range wireless communication such as WiFi; and (iii) infrastructure based communication such as 3G cellular. Via the infrastructure, a peer is able to transmit messages to another peer by MMS (Multimedia Message Service) or TCP/IP communication. This is referred to as the cellular-channel or cellular communication. Each peer has a network-id that is used as its address for cellular communication, and this id is required in order to send a message to the peer via the cellular channel. The network-id can be a cell-phone number or an IP address. In addition, peers can communicate via the WiFi channel if they are within transmission range. Knowledge of the network id is not necessary for this purpose. With the cellular infrastructure, the time is automatically synchronized among all the peers without any extra communication.

4.3.1.2 Reports and Reports Databases

Each peer periodically produces blob reports. Formally, a blob report R, is a couple <Meta(R), Blob(R)>, where Meta(R) and Blob(R) are the metadata and blob sub-reports, respectively. Meta(R) describes R using a resource description language such as RDF [64]. The metadata sub-report contains attributes describing the blob such as Time when the blob was produced, the Location at which it was produced, the Network-id of the producing peer, etc. Blob(R) is the blob itself, e.g., the music or video file.
A peer also produces queries that are stored and disseminated in the form of reports called *query-reports*. A query is represented using a resource query language such as SPARQL [65]. A query requests both sub-reports of each satisfying blob-report, but it refers only to the metadata of the blob-report. Thus, whether or not there is a match between a query-report and a blob-report can be determined solely based on the query and the metadata of the blob. For example, a query requests a song by its title, thus the match between the query and the song can be determined solely based on the metadata of the song-report, but the query asks for both the metadata and the song itself to be returned.

A peer is called the *producer* of the query- and blob-reports that it produces. Each query and each metadata sub-report contains the network-id of the producer of the report. A query requests all the blob reports that satisfy the query, independently of whether or not these reports reside with the query producer when the query is generated. However, the number of peers in the system and their network-id’s are unknown by a peer, and peers may be unreachable. Therefore, each query $Q$ has a *query expiration time*, which indicates that the processing of $Q$ stops when its expiration time is reached. In other words, the system delivers to the query originator the answers that it can obtain before the query expiration time, and then quits. In this sense query processing in a mobile P2P system is “best effort”, since delivery of all the answers is not guaranteed.

Each peer maintains a reports database that consists of three relations, namely a metadata-reports relation (MRR), a blob-reports relation (BRR), and a query-reports relation (QRR). These relations store the metadata sub-reports, blob reports, and query reports produced by the peer or received from other peers. Each tuple in the blob-reports relation contains both, the metadata and the blob of a report, whereas a tuple in the metadata reports relation contains only the metadata of such a blob-report. If a match between the metadata and a query is found, then the network-id in the metadata can be used to access the blob. The reason for maintaining separate relations for metadata and blob reports is that metadata may be disseminated separately from its blob, as will be explained in the next subsection.
4.3.1.3 WiFi Communication

WiFi communication at a peer is implemented by a WiFi Communication Module (WCM) that is invoked by the query processing strategy executing at the peer. Each invocation provides the WiFi communication method with a set of reports to be transmitted. As a consequence, at any point in time the WCM stores a transmission set, i.e. reports to be transmitted. Each transmission sends a subset of the transmission set to all the current neighbors, i.e. all peers that are currently within transmission range.

Addition of a report to the transmission set is an idempotent operation, i.e. adding the report again after its first addition does not change anything. The WCM may service multiple applications, and thus the WCM transmits the reports in the transmission set with a frequency that will optimize communication for all the applications. Furthermore, due to bandwidth limitations a transmission may not be able to send all the reports in the transmission set, thus the reports are prioritized by the WCM. The transmission frequency and priority are outside the control of the query processing strategy, and are executed by the WCM such that the WiFi communication efficiency is maximized.

Due to WiFi communication errors and limited bandwidth, it is more likely that long reports (i.e. blobs) get lost or delayed than short ones (i.e. queries and metadata sub-reports). Thus a short report may propagate differently from a long one, even if initially they are both broadcast simultaneously from the same peer.

4.3.2 The WiMaC Query Processing Strategies

As mentioned above, some or all reports that satisfy a query \( Q \) may reside on peers that are different than the query producer, \( Q_p \). Since \( Q_p \) does not normally have the network id of such peers, and does not even know how many reports satisfy the query, all query processing strategies start with a stage of WiFi dissemination to neighboring peers. The dissemination may be of the query, the blob reports, the metadata sub-reports, or some combination. When a match is found, it may be followed by a second stage of additional cellular or WiFi communication. For example, assume that the match is between a query and a metadata report, and that the blob sub-report is located at another peer. Then the blob has to be transferred
to the query producer by additional communication. Thus, this is the (WiFi-communication, Match, Communication) paradigm, called WiMaC, and all query processing strategies discussed in this thesis are special cases of WiMaC.

The query processing strategy operating at a peer is not concerned with communication issues such as the location, distance, or direction of the destination peer of a report. If necessary, these are handled by the communication layer.

This section discusses the WiMac strategies, and is organized as follows. In §4.3.2.1 we present the structure of the design space, resulting in 13 WiMaC query processing strategies. In §4.3.2.2 we prove that four strategies dominate the others.

4.3.2.1 Strategies Design-Space: An Overview

The structure of the design space is depicted in Table V, and is explained as follows. As aforementioned, the first stage of the WiMaC paradigm is a WiFi dissemination which finds a match. The WiFi dissemination is possibly followed by a second stage to complete query processing. There are seven design choices for the first stage, depending on whether the query, the blob reports, the metadata sub-reports, or some combination thereof are disseminated (see left column of Table V which indicates the disseminated combination).

In the (blob, meta) and (blob, meta, Q) choices, blob reports and metadata sub-reports are disseminated independently. Observe that a blob report contains its metadata sub-report, and the reason for disseminating the metadata sub-report alone in addition to the blob report is as follows. As discussed in sec. 4.3.1.3, blob reports and metadata sub-reports propagate differently by WiFi; the metadata report propagates faster, thus possibly meets more queries within a given time period.

The design choices for the second stage depend on whether the second stage follows at all, and if so whether WiFi or cellular communication is used for this stage (see right column of Table V). Observe that when a query and matching metadata report meet at a peer V, then the network id’s of both the query
producer and the report producer are known to V since they are in the query and metadata report, respectively. Thus the second stage can be conducted via the cellular infrastructure.

Observe that if only blob reports are disseminated at the first stage (i.e., the (blob) choice), then there is no need for the second stage. The reason is that in this case the query is not disseminated thus the match must have been found at the query producer, with the blob report constituting the answer.

Even if the strategy has a second stage, it does not mean that the second stage always executes after a match at a peer. For example, for the (blob, meta) case both blob and metadata reports are disseminated in the first stage of WiMaC. If a match involving a metadata report occurs, then a second stage is necessary to get the actual blob report. But if the match involves a blob report, then the second stage is not necessary because the blob report is already at the query producer. Similarly, for the (blob, Q), and (blob, meta, Q) cases, a second stage may not occur after a match at a peer.

**Table V. Design space of the WiMaC paradigm**

<table>
<thead>
<tr>
<th>No.</th>
<th>Type of reports disseminated in the first stage (always via WiFi)</th>
<th>Communication medium in the second stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(blob)</td>
<td>No second stage</td>
</tr>
<tr>
<td>2</td>
<td>(meta)</td>
<td>WiFi (2a), cell (2b)</td>
</tr>
<tr>
<td>3</td>
<td>(Q)</td>
<td>WiFi (3a), cell (3b)</td>
</tr>
<tr>
<td>4</td>
<td>(blob, meta)</td>
<td>WiFi (4a), cell (4b)</td>
</tr>
<tr>
<td>5</td>
<td>(blob, Q)</td>
<td>WiFi (5a), cell (5b)</td>
</tr>
<tr>
<td>6</td>
<td>(meta, Q)</td>
<td>WiFi (6a), cell (6b)</td>
</tr>
<tr>
<td>7</td>
<td>(blob, meta, Q)</td>
<td>WiFi (7a), cell (7b)</td>
</tr>
</tbody>
</table>

**Notation:**
blob = blob-report, meta = metadata sub-report, Q = query, cell = cellular.

**Strategy Names:** 1 is (blob), 2a is (meta)-WiFi, 3b is (Q)-cell, etc.
Each strategy is denoted as follows. The denotation consists of the strategy number as defined in Table V and the strategy name. The strategy name is formed as follows. If there is not the second stage, then the strategy is named by the first stage, i.e., (blob). If there is a second stage, then the strategy is named by the two stages connected by a “-”. For example, 2b (meta)-cell denotes the 2b strategy which disseminates metadata sub-reports in the first stage and uses cellular communication in the second.

4.3.3 Strategy Dominance Analysis

Let a peer receive an answer blob report at time t. The response-time of the answer is the length of the time period since the answer is produced until t.

We say that a strategy X is dominated by another strategy Y if the following 4 conditions are satisfied for every blob report B:

1. For every query that B answers, if the answer is received in Y, it is also received in X;
2. For every query that B answers, its response-time in Y is no higher than that in X.
3. The WiFi communication cost of B in Y is not higher than that in X.
4. The cellular communication cost of B in Y is not higher than that in X.

Intuitively, if X is dominated by Y, then the performance and the efficiency of X are no better than those of Y and therefore X is not worth further studying. In this subsection we identify the strategies that are dominated.

In the dominance analysis, the communication cost (but not the delay) of query-reports and metadata sub-reports is ignored for WiFi communication. Similarly, the communication cost of these reports is ignored for cellular communication. This is because query-reports and metadata sub-reports are very short. However, the simulations take into account the communication overhead of the query-reports and metadata sub-reports.

We say that strategy X is weakly dominated by strategy Y if the above dominance relationship only satisfies conditions 1-3, i.e. the cellular communication cost of Y may be higher. Weak dominance is appropriate for unlimited data plans offered by some cellular service providers.
The dominance relationship is summarized in Figure 20. Intuitively strategies 1 and 3a are incomparable because 3a disseminates only blobs that answer queries, whereas 1 disseminates all blobs thus its communication cost is higher; on the other hand, since 1 disseminates all blobs as soon as they are produced, its response time is lower. Similarly, 7b and 6b are incomparable because the WiFi communication cost of 7b is higher, but its response time may be lower.

Observe that each dominated strategy is dominated by a strategy from the same group and thus is not worth further studying. Thus the next section focuses on the non-dominated strategies.

Figure 20. Dominance relationship among strategies.

Description of the four non-dominated strategies

1 (blob): In this strategy, in the first stage of WiMaC blob reports are disseminated via WiFi (see Figure 21). In other words, blob reports are simply pushed by WiFi. Queries are kept at the producer peer,
and a match occurs when a disseminated blob report arrives at a matching query. There is no second stage.

Figure 21. Illustration of the 1 (blob) strategy.

3a (Q)-WiFi: In this strategy, in the first stage of WiMaC queries are disseminated via WiFi (See Figure 22). When a query Q reaches the producer peer of a matching blob report M, the B-producer disseminates the blob report via WiFi to reach the Q-producer.

Figure 22. Illustration of the 3a (Q)-WiFi strategy.
**6b (meta,Q)-cell:** In this strategy metadata and query reports are disseminated via WiFi in the first stage (see Figure 23). When a metadata sub-report B and matching query Q collocate at a peer Z, Z sends Q to the B-producer via the cellular channel. In response, the B-producer sends the blob report B and all the other matching blob reports that it has to the Q-producer, via the cellular channel.

![Figure 23. Illustration of the 6b (meta,Q)-cell strategy.](image)

**7b (blob,meta,Q)-cell:** This strategy is a combination of 5b (blob,Q)-cell and 6b (meta,Q)-cell in the sense that it does everything that (blob,Q)-cell does, and also everything that (meta,Q)-cell does.
4.3.4 **Comparison of Non-dominated Strategies by Theoretical Analysis**

In this section we compare the two non-dominated strategies, namely 1 (blob) and 6b (meta,Q)-cell, by theoretical analysis. For the performance measure we consider the delivery ratio, namely the fraction of answers that are received by a query before the query expires. The delivery ratio measures a strategy’s capability of delivering answers to queries. We provide an analytical model that computes the delivery ratio as a function of three parameters, including the number of peers in the network, the intensity with which they encounter each other, and the lifetime of queries. In §4.3.4.1 we introduce some assumptions in addition to the ones made in §4.3.1. In §4.3.4.2 we develop the analytical model. In §4.3.4.3 we compare 1 and 6b using the analytical model.

**4.3.4.1 Additional Assumptions**

For the analysis of this section we make the following assumptions.

1. A query Q expires T time units after it is produced, and requests the blob reports that are produced after Q. Remember that, as defined in §4.3.1, queries may refer to the blob metadata in an arbitrary way, Thus, the query may place additional restrictions, e.g. LocationOfBlobProduction=(x, y); in other words, the results of this section still apply even if additional constraints are imposed in the query.

2. The network consists of a fixed set of N+1 peers. As defined in §4.3.1.1, the set of peers may change over time. So again, this is a restriction.

3. An *encounter* is an event in which the distance between two peers A and B was larger and becomes smaller than the WiFi transmission range. When a peer O encounters another peer O’, in the 1 (blob) strategy, O’ receives from O all the blob reports that O stores, and vice versa; in the 6b (meta-Q)-cell strategy, O’ receives from O all the metadata reports and the query reports that O stores, and vice versa. This exchange succeeds with probability 1, and constitutes the only communication between peers until the next encounter. In the literature, this “forward-upon-encounter” paradigm is known as *epidemic dissemination* (see e.g., [66, 67, 68]).
4. Any pair of peers encounter each other by a Poisson process with intensity $\lambda$. In other words, expectedly each pair of peers encounter each other $\lambda$ times per time unit. This assumption is supported by the results of [69], which show that the encounters between a pair of peers follow a Poisson distribution if the peers move in a limited region according to common mobility models (such as the random waypoint or the random direction model).

5. The cellular communication is always successful. The delay for transmitting a query report via the cellular channel is a constant $c$. The delay for transmitting a blob report via the cellular channel is a constant $b$.

4.3.4.2 Analytical Model

In this subsection, we provide an analytical model that computes the delivery ratio. We first give the formal definition of the delivery ratio. Then in §4.2.1 we analyze the delivery ratio of strategy 1 (blob). In §4.2.2 we analyze the delivery ratio of strategy 4.2.2 6b (meta,Q)-cell.

Intuitively, the delivery ratio is the average ratio between the number of answers received by Q (obviously before Q’s expiration time) and the total number of answers produced in the system. More precisely, given an arbitrary query Q and an arbitrary answer R that is produced $t$ ($0 \leq t \leq T$) time units after Q, denote by $F(t)$ the probability that R is received as an answer to Q. The delivery ratio is defined to be the expected value of $F(t)$ when $t$ is uniformly distributed between 0 and T. In other words, if $t$ is a uniformly distributed random variable, then $F(t)$ is a random variable whose expected value is the delivery ratio. For example, if on average an answer is received by Q with 90% probability, then 90% of the answers are received by Q, and therefore the delivery ratio is 90%.

4.3.4.2.1 Delivery Ratio of Strategy 1 (blob)

Theorem 4.2. The delivery ratio of the 1 (blob) strategy is

$$\frac{\ln(e^{\lambda N T} + N - 1) - \ln N - \lambda \cdot T}{\lambda \cdot (N - 1) \cdot T}$$

(4.3.1) □
The proof of Theorem 4.2 is based on a known result in the literature of epidemic dissemination (see [66, 67, 29]). This result, stated by Lemma 4.1 below, gives the number of copies of a report in the system as a function of the report’s age.

**Lemma 4.1:** Let X be a query, blob, or metadata report. Denote by \( P(\tau) \) the probability that an arbitrary peer other than X-producer has X within \( \tau \) time units after X is produced. Then:

\[
P(t) = 1 - \frac{N}{N - 1 + e^{\lambda N \tau}} \tag{4.3.2}
\]

where \( 0 \leq \tau \leq T \).

**Proof:** Lemma 4.1 has been introduced as Theorem 1 in [29] and proved by obtaining a differential equation. □

Now we prove Theorem 4.2.

**Proof of Theorem 4.2:** Let a query report Q be produced at a peer Q-producer at time 0. Let a blob report B be produced at another peer B-producer at time \( t \) \((0 \leq t \leq T)\). Denote by \( F_1(t) \) the probability that the Q-producer receives B by time \( T \) (i.e., \( T-t \) time units after B is produced) using strategy 1. Clearly, \( F_1(t) = P(T-t) \). The delivery ratio of 1 (blob) is the expected value of \( F_1(t) \). Based on Lemma 4.1, the expected value of \( F_1(t) \) is:

\[
E[F_1(t)] = \frac{\int_0^T F_1(t)dt}{T} = \frac{1}{T} \int_0^T P(T-t)dt = \frac{1}{T} \int_0^T (1 - \frac{N}{N - 1 + e^{\lambda N (T-t)}}) dt = \frac{\ln(e^{\lambda N T} + N - 1) - \ln N - \lambda \cdot T}{\lambda \cdot (N-1) \cdot T}
\]

□

**4.3.4.2.2 Delivery Ratio of Strategy 6b (meta,Q)-cell**

**Theorem 4.3.** The delivery ratio of the 6b (meta,Q)-cell strategy is
\[
\frac{1}{T} \cdot (P(T-b) \cdot c + \int_0^{T-b-c} (1 - (1 - P(T-b)) \cdot (1 - P(T-b-c-t)) \cdot (1 - P(T-b-c) \cdot P(T-b-c-t))^n) \, dt)
\]

(4.3.4)

where the function P(x) is defined in Eq. 4.3.2, and b and c are defined in §4.3.4.1.

The proof of Theorem 4.3 is based on the following Lemma.

**Lemma 4.2**: Let a query report Q be produced at a peer Q-producer at time 0. Let a blob report B and its metadata report M be produced at another peer B-producer at time \( t \) (\( 0 \leq t \leq T \)). Denote by \( F_{6b}(t) \) the probability that the Q-producer receives B by time T.

If \( t \leq T - b - c \), then

\[
F_{6b}(t) = 1 - (1 - P(T-b)) \cdot (1 - P(T-b-c-t)) \cdot (1 - P(T-b-c) \cdot P(T-b-c-t))^n. 
\]

If \( T - b - c < t \leq T - b \), then \( F_{6b}(t) = P(T - b) \).

If \( t > T - b \), then \( F_{6b}(t) = 0 \).

**Proof**: If \( t > T - b \), then the Q-producer cannot receive B by time T because the delay of transmitting B via the cellular channel is b. Thus if \( t > T - b \) then \( F_{6b}(t) = 0 \).

Next we analyze the case of \( T - b - c < t \leq T - b \). In this case, the Q-producer receives B by time T if and only if the B-producer has Q by time T−b. The reason is that if and only if the B-producer receives Q by T−b, then it finishes transmitting B to the Q-producer by T since the delay of the transmission of B is b. Thus, if \( T - b - c < t \leq T - b \), then \( F_{6b}(t) = P(T - b) \).

Now we analyze the case of \( t \leq T - b - c \). In this case, the Q-producer receives B by time T if and only if at least one of the following three events occurs:

The B-producer has Q by T−b. If this event occurs, then the B-producer transmits B to the Q-producer, which takes b time units. Thus the Q-producer receives B by T. Event (i) means that the B-producer has Q within T−b time units after Q is produced, the probability of which is \( P(T - b) \).

The Q-producer has M by T−b−c. If this event occurs, then the Q-producer sends Q to the B-producer, which takes c time units. The B-producer then transmits B to the Q-producer, which takes b time units.
Thus Q receives B by T. Event (ii) means that the Q-producer has M within T−b−c−t time units after M is produced (Recall that M is produced at t). Thus the probability for event (ii) to occur is \(P(T−b−c−t)\).

There exists at least one peer O such that O is neither the Q-producer nor the B-producer and it has both Q and M by T−b. If this event occurs, then O sends Q to the B-producer, which takes c time units. The B-producer then transmits B to the Q-producer, which takes b time units. Thus Q receives B by T. O has both Q and M by T−b means that it has Q within T−b−c time units after Q is produced and it has M within T−b−c−t time units after M is produced. Thus the probability that O has both Q and M by T−b is \(P(T−b)\)⋅\(P(T−b−c−t)\). Since there are N−1 peers that are neither the Q-producer nor the B-producer, the probability that none of them has both Q and M by T−b−c is \(1−(1−P(T−b)\cdot P(T−b−c−t))^{N−1}\). Thus, the probability for event (iii) to occur is \(1−(1−P(T−b−c)\cdot P(T−b−c−t))^{N−1}\).

Thus, the probability that at least one of the above three events occurs is

\[
1−\text{Prob(none of the three events occurs)}=1−(1−P(T−b))\cdot (1−P(T−b−c−t))\cdot (1−P(T−b−c)\cdot P(T−b−c−t))^{N−1}\]

Now we are ready to prove Theorem 4.3.

**Proof of Theorem 4.3.** The delivery ratio is the expected value of \(F_{6b}(t)\). Based on Lemma 4.2, the expected value of \(F_{6b}(t)\) is

\[
E[F_{6b}(t)] = \int_0^T F_{6b}(t)dt \quad T
\]

\[
= \frac{1}{T} \int_0^{T−b−c} F_{6b}(t)dt + \int_{T−b−c}^{T} F_{6b}(t)dt + \int_{T−b}^{T} F_{6b}(t)dt
\]

\[
= \frac{1}{T} \int_0^{T−b−c} (1−(1−P(T−b))\cdot (1−P(T−b−c−t))\cdot P(T−b−c−t))^{N−1}dt + \int_{T−b−c}^{T} P(T−b)dt
\]

\[
= \frac{1}{T} \cdot (P(T−b)\cdot c + \int_0^{T−b−c} (1−(1−P(T−b))\cdot (1−P(T−b−c−t))\cdot P(T−b−c−t))^{N−1}dt)
\]

**4.3.4.2.3 Comparison between 1 (blob) and 6b (meta-Q-cell)**
In this subsection we compare the 1 (blob) strategy and the 6b (meta-Q)-cell strategy using Theorem 4.2 and Theorem 4.3. Specifically, we replace the parameters in Eq. 4.3.1 and Eq. 4.3.4 by specific values to compute the delivery ratio of strategies 1 (blob) and 6b (meta-Q)-cell, respectively. Out of these parameters, b and c are fixed; N+1, λ, and T are variable. We generate three figures where each figure plots the delivery ratio as a function of one of the three variables (N+1, λ, and T).

The parameters b and c are setup based on the transmission speed of HSPDA, a currently available 3G cellular technology. The transmission speed of HSPDA is 384Kbps. The size of a query is assumed to be 1K bytes; the size of a blob report is assumed to be 1M bytes. Thus c=0.02 second and b=20 seconds.

All the parameters and their values are specified in Table VI.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of peers</td>
<td>N+1</td>
<td>10 ~ 1000</td>
</tr>
<tr>
<td>Encounter intensity</td>
<td>λ</td>
<td>0.0001 ~ 0.003 sec⁻¹</td>
</tr>
<tr>
<td>Query lifetime</td>
<td>T</td>
<td>1 ~ 300 seconds</td>
</tr>
<tr>
<td>Delay for transmitting a query report</td>
<td>c</td>
<td>0.02 second</td>
</tr>
<tr>
<td>via the cellular channel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delay for transmitting a blob report</td>
<td>b</td>
<td>20 seconds</td>
</tr>
<tr>
<td>via the cellular channel</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Figure 24. Delivery ratio versus encounter intensity, with N+1=100 and T=300 seconds.

Figure 25. Delivery ratio versus query lifetime, with N+1=100 and \( \lambda=0.001 \text{ sec}^{-1} \).
Figures 24, 25, and 26 show the comparison between 6b and 1 under various parameter configurations. From these figures it can be seen that for a wide range of the delivery ratio, 6b is much better than 1. 1 is better than 6b only when the delivery ratio is either very high (close to 1, see Figure 24 and Figure 26) or very low (close to 0, see Figure 25). Practically, this means that 6b is a better strategy than 1, because when the delivery ratio is close to 1 or 0 (where 1 is better than 6b), it does not matter much which strategy is better.

Intuitively, the performance of strategy 6b (meta,Q)-cell is limited by the delay of the cellular communication. Thus, even with a high encounter intensity or a large number of peers, the delivery ratio of strategy 6b is limited to a certain (high) level (see Figure 24 and Figure 26). The performance of strategy 1, on the other hand, constantly increases with the encounter intensity and the number of peers. However, the performance of 6b increases much faster than 1. Practically, this means that compared to strategy 1 strategy 6b requires a much lower network density to reach a reasonable performance. For example, according to Figure 26 strategy 1 requires 600 peers in order for the delivery ratio to reach 0.9, whereas strategy 6b only requires 35 peers to do so.
4.3.5 **Comparison of Non-dominated Strategies by Simulations**

In this section we compare by simulation the four non-dominated query processing strategies, namely: 1 (blob), 3a (Q)-WiFi, 7b (blob,meta,Q)-cell, and 6b (meta,Q)-cell (see Table VI). To put results in perspective, we also compare the WiMaC strategies with an ideal benchmark which represents the performance of a perfect client/server system. The comparisons are based on the application of delivering traffic multimedia clips among moving vehicles in order to warn drivers about traffic jams and dangers. §4.3.5.1 elaborates the application scenario. §4.3.5.2 describes the simulation environment which attempts to mimic actual traffic conditions and communication details. §4.3.5.3 introduces the performance measure and defines the ideal benchmark. §4.3.5.4 presents the simulation results.

**4.3.5.1 Multimedia Traffic Information Application**

In this application, each vehicle periodically captures short (2-seconds) multimedia clips consisting of audio and/or video of the traffic conditions surrounding it. We have conducted a field experiment in which automatically captured videos from dashboard-mounted smart-phones are disseminated among vehicles in a peer-to-peer fashion, and the results can be seen at [70]. Each video clip is 2 seconds long with the size of 65K bytes. The traffic conditions can be readily discerned by the viewer (the receiving driver) of the video.\(^6\)

Each multimedia clip is encapsulated in a blob report R \(<\text{Blob(R), Meta(R)}\\rangle\). Meta(R), the metadata sub-report, is a 3-element tuple defined as follows.

\(<\text{producer, produce-time, produce-location}}\rangle\), where:

*producer*: the network-id of the vehicle that produced R

---

\(^6\) We understand the safety concerns that are raised by a video system that requires the driver attention. Our answer is threefold. First, the video viewing function is optional, so users can simply turn it off. Second, often traffic videos will be viewed while the driver is stuck in traffic (probably this is the reason s/he requested the video of the traffic ahead), thus stationary. Third, remember that the videos are very short; 2 seconds is sufficient to see whether or not the traffic is flowing, and at what speed. In fact often a picture is sufficient, and in this case glancing at a traffic picture is equivalent to glancing at the screen of a car-navigation device. Such devices have become popular, and have been shown to have few safety concerns.
*produce-time*: the time when R starts to be captured

*produce-location*: the (x,y) location where R starts to be captured

Blob(R), the blob sub-report, is the multimedia clip encapsulated in R. Based on experiments with a smart-phone video camera, its size in the simulations is taken to be 65K bytes.

A query report is a 4-element tuple defined as follows.

<producer, time, location, target-region>, where:

producer: The network-id of the query producer

time: The time at which the query is issued

location: The location at which the query is issued

*target-region*: A geographic region of interest. It indicates that the query producer is interested in receiving multimedia clips that started to be captured in this region.

Each blob report and each query report has a *lifetime* which defines the period of time during which it is considered valid. In the simulations all the reports have the same lifetime which is a system parameter. A report is dropped by the vehicles when its lifetime expires. A blob report R, or its metadata sub-report, *satisfies* a query report Q if (i) R is produced after the produce-time of Q; (ii) R.location falls within Q.target-region.

4.3.5.2 Simulation Environment
Figure 27. Simulated road network: portion of the highway system in a US city. Total road length is 96 km.

The simulation area is taken to be a section of the road network illustrated in Fig. 4.3.8, whose total length is 96 km. The network is obtained from TIGER/Line data (see [71]) for a US County. We used SWANS++ [72] as the simulation tool. SWANS++ is a network simulator that integrates vehicle mobility and WiFi communication. For mobility, SWANS++ provides a microscopic vehicular traffic model called STRAW [73]. In the STRAW model, vehicle movement is constrained to roadways defined by real maps. The vehicle mobility is limited according to the speed limit of each road segment, car-following rules (in which, for example, the average speed depends on the density of vehicles), and traffic control mechanisms (e.g., stop signs and traffic lights).

For WiFi communication we used two models. First is the SWANS++ model which implements the IEEE 802.11b Medium Access Control (MAC) protocol; it accounts for contention and collisions. The broadcast UDP (user datagram protocol) over IEEE 802.11 networks, which is part of SWANS++, is employed for WiFi communication. More specifically, the simulations adopt a stateless approach to the WiFi dissemination of reports. In this approach, the dissemination is conducted by each vehicle broadcasting reports to its neighbors. We do not assume route establishment or handshakes between senders and receivers. The dissemination is non-directional. In other words, all the neighbors of a sender
may rebroadcast the received report, independently of their locations between the sender and the receiver. We choose the non-directional dissemination because of the high mobility and the possibility of network partitions in our environment. We also use smart-flooding [74], and cooperative caching [29]. Since these are existing techniques we do not elaborate on them further.

We augmented SWANS++ with cellular communication based on the typical parameters of 3G communication (see [75] and [76]). Specifically, the simulated area is divided into 2.5km×2.5km squares (cells). In each cell, at most 30 users may transmit concurrently. The data transmission rate is 384 Kbps. That is, each vehicle may transmit at most 384 Kbits per second via the cellular channel.

We realize that the simulation results may heavily depend on the used network simulator (SWANS++ in our case), as suggested in [77]. In order to test the robustness of the results obtained from the SWANS++ communication model, we conducted simulations with the communication component of SWANS++ replaced by a simple communication model. The simple model does not simulate the detailed procedure of WiFi channel contention and collisions. Instead, it takes into account the bandwidth constraint as follows. When a vehicle O makes a transmission, the transmission is received by all its neighbors reliably and instantaneously. However, the size of the transmission, i.e., the number of bytes included in the transmission, is limited based on the WiFi bandwidth, the number of neighbors of O, and the inter-transmission time interval. Specifically, let $G$ be the WiFi bandwidth, $M$ the number of vehicles with the transmission range of O including O itself, and $H$ the length of the time period starting from the last transmission by O until the current transmission time. Then the transmission size is limited to be

$$\frac{G}{M} \cdot H$$

Intuitively, $\frac{G}{M}$ is the maximum number of bytes that can be transmitted by O per time unit under the condition that the WiFi bandwidth is equally shared by O and its neighbors. $\frac{G}{M} \cdot H$ is the transmission size accumulated from the last transmission until the current one.

---

7 The authors in [46] discover that due to mismatching communication models, significant differences exist between different network simulators for the same simulated scenario.
Table VII lists all the simulation parameters. We tested two traffic scenarios, namely light-congestion (4000 vehicles), and heavy-congestion (8000 vehicles with road construction on 50% of randomly selected road segments).

For each traffic scenario, only a fraction of the entire vehicle population generates multimedia clips and participates in the WiMac query processing. This fraction is referred to as the penetration ratio. The penetration ratio varies from 1% to 50% for each traffic scenario. By varying the penetration ratio we varied the density of the WiFi network. The mapping between the penetration ratio and the average inter-vehicle distance is given in Table VIII.

The query target region is at distance 1600 meters, and has a width of 500 meters. This means that a vehicle is interested in traffic multimedia clips captured in the area lying between 1350 meters and 1850 meters away from the current query producer’s location.

Every 10 seconds, each vehicle produces a blob report with a probability that is in reverse proportion to the penetration ratio. It reflects the realistic fact that not all participating (in WiMac) vehicles are producing reports. Thus, in our simulations the density of participating peers varies, but the supply of blob reports is fixed. Particularly, the supply of blob reports is fixed at 4 per second. Queries are produced as follows. Every 300 seconds, each participating vehicle produces a query with a probability called the query ratio. The query ratio is a system parameter. Thus, for example, if the query ratio is 25%, then when the penetration ratio is 4%, every 300 seconds 1% of the vehicles produce queries.
Table VII. Simulation parameters and their values

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total length of road segments, total simulated area.</td>
<td>96 km, 24×31 sq. km. (see fig. 4.3.8)</td>
</tr>
<tr>
<td>Traffic condition</td>
<td>Light-congestion: 4000 vehicles, 64km/hour avg speed over time among all road segments</td>
</tr>
<tr>
<td></td>
<td>Heavy-congestion: 8000 vehicles, reduced speed-limit for 50% of road segments, such that avg speed over time among all road segments is 25km/hour</td>
</tr>
<tr>
<td>Penetration ratio ($P_{ratio}$)</td>
<td>1% ~ 50%</td>
</tr>
<tr>
<td>WiFi transmission range, data transmission rate of WiFi channel</td>
<td>250 meters, 2 Mbps(^8)</td>
</tr>
<tr>
<td>Side-length, capacity of each cell</td>
<td>2.5 km, 30 users</td>
</tr>
<tr>
<td>Data transmission rate of cellular channel</td>
<td>384 Kbps</td>
</tr>
<tr>
<td>Mobility model</td>
<td>STRAW</td>
</tr>
<tr>
<td>WiFi communication model</td>
<td>SWANS++, simple</td>
</tr>
<tr>
<td>Frequencies of query generation</td>
<td>Every 300 seconds with randomization</td>
</tr>
<tr>
<td>Query ratio</td>
<td>0.25, 0.5, 0.75, 1</td>
</tr>
<tr>
<td>Frequency of blob report generation</td>
<td>Every 10 seconds with randomization</td>
</tr>
<tr>
<td>Query distance, query width</td>
<td>1600 meters, 500 meters.</td>
</tr>
<tr>
<td>Query/clip report lifetime</td>
<td>60, 120, 180, 240, 300 seconds</td>
</tr>
<tr>
<td>Sizes of query, metadata, and multimedia-clip</td>
<td>40bytes, 28bytes, and 65Kbytes, respectively.</td>
</tr>
<tr>
<td>Reports database size</td>
<td>6 Mbytes</td>
</tr>
<tr>
<td>Length of a simulation run</td>
<td>3600 simulated seconds</td>
</tr>
</tbody>
</table>

\(^8\) 2Mpbs is the max transmission rate, i.e., without contention and collisions. Contention and collisions are accounted for by the simulation system, and they reduce the bandwidth according to the density of vehicles.
Table VIII. Mapping between the penetration ratio and the average distance between two neighboring participating vehicles. (distance unit: meter)

<table>
<thead>
<tr>
<th></th>
<th>1%</th>
<th>5%</th>
<th>12.5%</th>
<th>25%</th>
<th>37.5%</th>
<th>50%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Light congestion</td>
<td>2400</td>
<td>480</td>
<td>192</td>
<td>96</td>
<td>64</td>
<td>48</td>
</tr>
<tr>
<td>Heavy congestion</td>
<td>1200</td>
<td>240</td>
<td>96</td>
<td>48</td>
<td>32</td>
<td>24</td>
</tr>
</tbody>
</table>

4.3.5.3 Performance Measures

For each strategy we evaluated the following performance parameters:

**Answer throughput:** The answer throughput is the number of distinct answers (i.e., matching blob reports) received for each query. An answer is counted towards the throughput only if both the answer and the query have not expired at the time when the answer is received.

In order to put the performance of the WiMaC strategies in perspective, we define an ideal benchmark algorithm that delivers reports instantaneously. Thus, after a vehicle produces a query, it instantaneously receives every answer that is generated during the query’s life span. This resembles an installation which has a high performance centralized database and numerous WiFi hotspots (through which these databases are accessed from everywhere).

**Communication overhead:** The average number of bytes per vehicle submitted to the MAC level during the simulation, by the WiFi channel and the cellular channel, respectively. In other words, this overhead is the amount of attempted communication, the amount communicated successfully is lower.

4.3.5.4 Simulation Results

In §4.3.5.4.1 we present the results with regard to the answer throughput measure. In §4.3.5.4.2 we present the results with regard to the communication overhead measure. The results presented in the above two subsections were obtained using the SWANS++ communication model. In §4.3.5.4.3 we present the results obtained using the simple communication model.
4.3.5.4.1 Answer throughput

Figures 28 and 29 show the answer throughput as a function of the penetration ratio, for the light-congestion and heavy-congestion scenarios, respectively. Figures 30 and 31 show the answer throughput as a function of the report lifetime. Figures 32 and 33 show the answer throughput as a function of the query ratio. It can be seen that in all the figures, ranking of the strategies based on throughput is 6b (meta,Q)-cell > 7b (blob,meta,Q)-cell > 1 (blob) > 3a (Q)-WiFi.

Best strategy. Strategy 6b (meta,Q)-cell is the clear winner. The advantage of 6b increases as the penetration ratio increases. In some cases, the answer throughput of 6b is seven times higher than those of the other strategies. Furthermore, the fact that 6b is much better than 1 (blob) is consistent with the analytical result in §4.3.4.

It is surprising that strategy 7b (blob,meta,Q)-cell is much worse than 6b. Compared to 6b, strategy 7b also disseminates blob reports via WiFi in the first WiMaC stage, thus vehicles have a chance to receive answers from the WiFi dissemination directly. Presumably the performance of 7b should be close to if not higher than that 6b. The poor performance of 7b is probably due to the fact that the WiFi dissemination of multimedia reports occupies a lot of WiFi bandwidth, which creates contention and collisions in the dissemination of metadata sub-reports and query reports. This interference significantly slows down the discovery of matches.
Figure 28. Answer throughput versus penetration ratio, light-congestion, SWANS++, report lifetime = 300 seconds, query ratio = 0.25.

Figure 29. Answer throughput versus penetration ratio, heavy-congestion, SWANS++, report lifetime = 300 seconds, query ratio = 0.25.
Figure 30. Answer throughput versus report lifetime, light-congestion, SWANS++, penetration ratio = 0.05, query ratio = 0.25.

Figure 31. Answer throughput versus report lifetime, heavy-congestion, SWANS++, penetration ratio = 0.05, query ratio = 0.25.
Comparison of WiFi-only strategies. 1 (blob) is better than 3a (Q)-WiFi. 1 and 3a represent two paradigms of query processing, i.e., 1 represents push (data-to-query) and 3a represents pull (query-to-
data). The simulation results show that push is better than pull for the considered environment. Intuitively, the pull strategy requires a round-trip dissemination in order for a query originator to receive an answer: the query has to travel from the query originator to the answer producer and then the answer has to travel back from the answer producer to the query originator. If either way does not go through or experiences a long delay, then the answer does not reach the query originator within the lifetime; and this scenario is likely in a highly mobile environment.

**Feasibility of WiMaC.** Even in the 1% penetration ratio and light congestion, the answer throughputs of 1, 6b, and 7b are at least 2 (see Figure 28). This fact is surprising, because when the penetration ratio is 1%, the inter-participating-vehicle distance is approximately 2400 meters (see Table VIII), i.e. much higher than the WiFi transmission range of 250 meters. With this density, the network is highly disconnected. In other words, it is expected that most of the time a vehicle does not have any neighbors within its transmission range. Yet on average each query receives at least one answer. This is due to the store-and-forward (namely cooperative caching) mechanism, which enables WiFi dissemination even when the network is highly disconnected.

In comparison with the ideal benchmark, in most cases, the answer throughput of the best strategy 6b is at least one-third of the ideal benchmark. When the penetration ratio is 12.5%, the answer throughput of 6b is 70% of the ideal benchmark in the light-congestion scenario (Figure 28) and 80% of the ideal benchmark in the heavy-congestion scenario (Figure 29).

**The Ideal Benchmark** lines in Figures 28, 10, 32, 33 fluctuate slightly as a function of the penetration ratio $p$ or the query ratio $q$, because the total number of queries generated grows with $p$ and $q$. Since a report satisfies multiple queries, the fluctuation is mild. Observe that in Fig. 4.3.10, the ideal benchmark line is lower than that in §4.3.9. The same is true for Fig. 4.3.12 versus 4.3.11, and for 4.3.14 versus 4.3.13. In other words, when the other parameters are fixed, the ideal benchmark of the heavy-congestion scenario is higher than that of the light-congestion scenario. The reason for this might be that in the heavy-congestion scenario, vehicles are clustered due to congestions. In this case, fewer multimedia
clips fall into the range of a query than in the light-congestion scenario where vehicles are more uniformly distributed.

**Impact of the penetration ratio (Figures 28 and 29).** When the penetration ratio increases, the answer throughputs of 1, 7b, and 6b initially increase, and then decrease. Intuitively, when the penetration ratio increases, two effects are generated:

The WiFi network becomes more connected, which pulls the answer throughput up;

The contention and collisions increase for both the WiFi network and the cellular network, which pulls the answer throughput down.

The answer throughput curves shown in Figures 28 and 29 are the result of the interplay of these two effects.

**Impact of the report lifetime (Figures 30 and 31).** Not surprisingly, for all the strategies, the throughput increases with the report lifetime.

**Impact of the query ratio (Figures 32 and 33).** The throughputs of 1 (blob) and 3a (Q)-WiFi do not change with the query ratio. This is because these strategies use WiFi broadcasting to disseminate blob reports, and each broadcast satisfies multiple queries. The throughput of 6b (meta,Q)-cell decreases with the query ratio. The reason is as follows. 6b uses the cellular channel to transmit blob reports. Each transmission satisfies exactly one query. Because of this, when there are more queries, the contention on the cellular channel increases, and therefore more queries are not satisfied.

4.3.5.4.2 Communication Overhead

Figure 34 shows the communication overhead as a function of the penetration ratio for the heavy-congestion scenario. The throughput for the same configuration is presented in Figure 29. It can be seen that the winning strategy in terms of throughput, 6b (meta,Q)-cell, has the lowest WiFi communication volume, because only metadata reports and query reports (which are short) are disseminated via WiFi. 6b has the highest cellular communication volume. When the penetration ratio is low (12.5% or below), the communication overhead of 6b (including WiFi and cellular) is lower than those of the other three
strategies. When the penetration ratio is high (25% or above), the communication overhead of 6b is higher than those of 1 (media) and 7b (blob,meta,Q)-cell. In this case, the 6-fold increase in throughput comes with a higher communication cost.

![Graph showing communication overhead versus penetration ratio.](image)

**Figure 34. Communication overhead versus penetration ratio, heavy-congestion, SWANS++, penetration ratio = 25%, report lifetime = 300 seconds, query ratio = 0.25.**

### 4.3.5.4.3 Results with the Simple Communication model

Figures 35 and 36 show the results obtained with the simple communication model. Otherwise the parameter configurations of these two figures are the same as those of Figures 28 and 29 (which were obtained with the SWANS++ communication model), respectively. Compared with the SWANS++ results, the absolute value of the answer throughput changes, due to the change of the communication model. However, the shapes of each curve change little. More importantly, the order of the compared strategies does not change. It is still 6b (meta,Q)-cell > 7b (blob,meta,Q)-cell > 1 (blob) > 3a (Q)-WiFi. The fact that the conclusions drawn in the SWANS++ communication model carry over to the simple communication model demonstrates the robustness of the conclusion.
Figure 35. Answer throughput versus penetration ratio, light-congestion, simple, report lifetime = 300 seconds, query ratio = 0.25.

Figure 36. Answer throughput versus penetration ratio, heavy-congestion, simple, report lifetime = 300 seconds, query ratio = 0.25.
4.3.6 Relevant Work

Multimedia and P2P databases. Most multimedia work in the database community has been on content based retrieval. In contrast, this these addressed the issue of mobile P2P query processing. Our work is somewhat relevant to context-based multimedia retrieval (see e.g. [78]). In this type of retrieval, multimedia data is retrieved based on the context such as time and location in which they are created. However, existing work in this area assumes a central server model and concentrates on the matching algorithm.

Video streaming in VANET’s. The authors in [79] propose a vehicle-to-vehicle live video streaming architecture called V3. The architecture adopts a query-to-data paradigm for query processing and it uses only WiFi communication. Thus the query processing method in V3 is similar to the (Q)-WiFi strategy. However, the work in [79] does not compare the query-to-data paradigm with other WiFi-only strategies and with WiFi-cellular strategies.

PAVAN [80] is a vehicular video streaming system. It uses central servers for the collection and dissemination of metadata information, and uses WiFi inter-vehicle communication for video streaming. Thus the system relies on central servers whereas our system is purely peer-to-peer.

The authors in [81] study reliable multimedia delivery in VANET’s. The problem they focus on is how to encode the frames of a video clip communicated via WiFi, so that the receiver can recover from packet losses. Their method is orthogonal to ours, and could be used in the WiMaC system.

4.4 Application to KNN Queries

In this section we demonstrate the MARKET algorithm in the context of K-nearest-neighbors (KNN) queries, where the reports are the current locations of mobile sensors. Thus, the MARKET algorithm is specialized to process this specific spatial query. We compare the MARKET KNN performance with that of the in-network KNN algorithm given in [40] (called DIKNN). This section is organized as follows. In §4.4.1 we discuss the instantiation of MARKET for KNN query processing. In §4.4.2 we compare
MARKET and the DIKNN algorithm which is designed especially for KNN query processing in mobile sensor networks.

4.4.1 **KNN Query Processing in MARKET**

We consider a system where there is a single static sink. The sink is continuously interested in knowing the $k$ mobile peers that are closest to a static query point $q$. Thus the sink and every other peer have a query $Q$ which contains the coordinates of $q$. $Q$ is known to all the peers before the KNN query processing and therefore it is not transmitted during a P2P interaction. A report is produced by each mobile peer $O$ every one second. The report, referred to as the location report, contains the current timestamp, the current location of $O$, and the peer-id of $O$. We refer to the location included in a report $R$ as the home location of $R$.

For the computation of the demand in eq. 3.1, $Q(R)$, i.e., the degree to which a report $R$ satisfies the KNN query $Q$ is negatively correlated to the distance between the home location of $R$ and the query point $q$, and is $[0,1]$ normalized. For example, if the distance between the home location of $R$ and $q$ is 1.4 mile, and the maximum distance between $q$ and the home location of a report in the peer’s database is 2 miles, then the demand for $R$ is $(1-1.4/2=0.3)$.

The reports propagate via P2P to the sink, which is one of the peers. The sink answers the KNN queries based on the latest location of each peer maintained in the sink’s reports database.

4.4.2 **Comparison with DIKNN**

The **DIKNN Algorithm**. In the literature there is no algorithm for in-network processing of continuous KNN queries. Thus we use an instantaneous algorithm, DIKNN, and execute it repeatedly. In DIKNN, the query issued by the sink is geographically routed from the sink to the nearest neighbor of the query point $p$ by GPSR ([82]). The query is then disseminated to all the peers in a circular searching area centered at $p$. The size of the searching area is determined based on the density of the network, such that
the $k$ nearest neighbors are inside the searching area with a high probability. At the end of the dissemination, the aggregated query response is routed back to the sink.

**Simulation Method.** MARKET is implemented in SWANS (Scalable Wireless Ad-hoc Network Simulator) built at Cornell ([83]). 100 peers move within a $L \times L$ square area according to the *random way-point* mobility model with mean speed 1 mile/hour and mean pause time 180 seconds. In this setup, the average number of peers $D$ within a circle with radius 100 meters (i.e., the transmission range), is

$$D = 100 \times \frac{\pi \times 100 \times 100}{L^2}.$$  

We refer to $D$ as the *peer density*. $D$ is a system parameter which varies from 1 to 6. $L$ is chosen such that the peer density equals to $D$. The whole simulation runs for 8 simulated hours.

For DIKNN, we do not simulate its operation. Instead, we compute the upper bound of its query accuracy. Assume that the geographic routing is reliable and instantaneous, and the searching area contains all the K-nearest neighbors, thus the query is correctly delivered to them. Then with DIKNN, a correct KNN is returned to the sink if and only if there exists a path between this KNN and the sink at the time when the query is issued. For each query, we compute the fraction of the correct KNN’s for which there are paths between them and the sink, and take this fraction to be the accuracy of DIKNN.

The above simulation method favors MARKET in the sense that MARKET is designed for continuous queries and DIKNN is designed for instantaneous queries. On the other hand, it favors DIKNN in the follow senses. First, DIKNN is optimized for KNN query processing whereas MARKET is a general purpose query processing algorithm. Second, MARKET is compared with the upper bound accuracy of DIKNN.

The sink is located at the center of the northwest quadrant of the simulated square area. The query point is located at the center of the southeast quadrant. The query is issued every one second. For MARKET, this means that KNN’s are computed at the sink using the local database every 1 second. The local database has the last known location of each peer, but some of these locations may be outdated, thus the computed KNN may not be accurate. The accuracy of each KNN query is measured by the fraction of the correctly returned KNN’s.
The size of each report is 24 bytes, including 16 bytes for the coordinates of the location, 4 bytes for the timestamp, and 4 bytes for the peer-id. The size limit of each reports database is randomly chosen from [1200, 3600] bytes. Thus the average number of members in a reports database is $2400K/24=100$. The query database holds exactly one query which is the only query in the system. The size of the query is 16 bytes for the coordinates of the query point. All the simulation parameters and their values are listed in Table IX.

Table IX. Simulation parameters and their values

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Unit</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of mobile peers</td>
<td></td>
<td>100</td>
</tr>
<tr>
<td>Transmission range</td>
<td>meter</td>
<td>100</td>
</tr>
<tr>
<td>Peer density</td>
<td>Number of peers in a circle with radius $r$</td>
<td>1, 2, 3, 4, 5, 6</td>
</tr>
<tr>
<td>Data transmission speed</td>
<td>bits/second</td>
<td>2M</td>
</tr>
<tr>
<td>Report production rate</td>
<td>reports/second/peer</td>
<td>1</td>
</tr>
<tr>
<td>Report size range</td>
<td>byte</td>
<td>24</td>
</tr>
<tr>
<td>Query size</td>
<td>byte</td>
<td>16</td>
</tr>
<tr>
<td>Reports database size range</td>
<td>byte</td>
<td>[1200, 3600]</td>
</tr>
<tr>
<td>Demand database size</td>
<td>byte</td>
<td>16</td>
</tr>
<tr>
<td>Mobility model</td>
<td></td>
<td>Random way-point</td>
</tr>
<tr>
<td>Average motion speed</td>
<td>miles/hour</td>
<td>1</td>
</tr>
<tr>
<td>Average pause time</td>
<td>second</td>
<td>180</td>
</tr>
<tr>
<td>Simulation run time</td>
<td>hours</td>
<td>8</td>
</tr>
</tbody>
</table>

Simulation results. Figure 37 shows the accuracy of MARKET and DIKNN when $k$ is 10, for different peer densities. Figures 38 and 39 show the accuracy when peer density is 2 and 1 respectively, for different values of $k$. From these figures it can be seen that the accuracy of MARKET is by far higher
than that of DIKNN when the peer density is low. When the peer density is 1, the DIKNN algorithm completely fails, with 0% accuracy. In this case MARKET manages to provide up to 30% accuracy (see Figure 39). When the peer density is 2, MARKET outperforms DIKNN by an order of magnitude, regardless of the value of $k$. In this case the accuracy of MARKET reaches 60% for $k=20$ (see Figure 38).

The reason for the poor performance of DIKNN in a sparse network is that, with very high probability there does not exist a path between the sink and a KNN. When the sink and the KNN are disconnected, the query does not reach the KNN and the KNN is not collected in the query response. MARKET, on the other hand, does not need a contemporaneous path to collect results. This demonstrates the benefit of store-and-forward in a sparse network. When the peer density is higher than 5 (see Figure 37), DIKNN starts to outperform MARKET.

![Figure 37. Accuracy as a function of peer density with $k=10$. The accuracy is zero for DIKNN for peer densities 0.5 and 1.](image-url)
4.5 Pattern of Report Propagation

In §4.5.1 we study the propagation pattern of reports for temporal resources and in §4.5.2 we study the propagation pattern of reports for spatio-temporal resources.

4.5.1 Propagation Pattern of Reports for Temporal Resources

In this subsection we theoretically analyze how a report is propagated per time and per distance with the MARKET algorithm. We consider a special case for the relevance function, where the decay factor of...
distance $\beta$ is zero. Thus the relevance is purely a function of the age. This relevance function models the decay of the reports that are only specific to time, e.g. the Dow Jones Industrial Average at a particular time. In this section we first introduce the parameters and our assumptions, and then we develop a mathematical model that describes the propagation of a report. Finally we use the mathematical model to analyze the propagation of reports.

4.5.1.1 Parameters and Assumptions

Let $N$ be the total number of peers in the system. We assume that the value of $M$ is 1 for all the peers although our result can be extended to the general case where $M$ is more than 1. Each peer interacts with other peers by a Poisson process with intensity $\lambda$. We assume that the length of the valid duration of a resource is 0, and the wireless transmission range is small enough such that at most one peer can receive the report when it is produced. Observe that with this assumption, the age of a report is always 0 when it is acquired. Further observe that the report may still be relevant even after it is invalid. For example, although a Dow Jones Industrial Average report is invalid right after it is generated, it is still of interest for a period of time. For another example, even after occupied, a parking slot report is relevant because peers do not always know whether it is occupied. We consider only the reports that are received by a peer when they are produced. Such reports are generated within the system by a Poisson process with intensity $\mu$. Peers are randomly distributed in the space at any point in time, and therefore each peer is equally probable to receive each produced report. A newly generated report is sent to exactly one peer. Thus each peer receives newly generated reports according a Poisson process with the rate $\mu/N$. Finally we assume that each report exchange is finished instantaneously.

4.5.1.2 A Mathematical Model for Report Propagation

Let us define two variables: $q(t)$: The conditional probability that a peer has a report for a resource $R$ at time $t$ ($t>0$) given that $R$ is created at time 0. $g(t)$: The probability that at time $t$ ($t>0$) a report that is created after 0 is in the reports database of a peer. Now consider $q(t+\Delta t)$ which is the probability that at
time \( t+\Delta t \) a report \( R \) that is created at time 0 is in the reports database of a peer \( o \). Let \( \Delta t \) be small enough such that at most one report is generated in the system between \( t \) and \( t+\Delta t \) and \( o \) can interact with at most one peer during the same time interval. \( q(t+\Delta t) \) is the probability that one of the following mutually exclusive events happens:

1. \( o \) has \( R \) at time \( t \), and it does not acquire any new report between \( t \) and \( t+\Delta t \), and it does not interact with any peer between \( t \) and \( t+\Delta t \). The probability for this to happen is \( q(t) \cdot (1 - \lambda \cdot \Delta t) \cdot (1 - \frac{\mu}{N} \cdot \Delta t) \).

2. \( o \) has \( R \) at time \( t \), and it does not acquire any new report between \( t \) and \( t+\Delta t \), and it interacts with one peer \( o' \) between \( t \) and \( t+\Delta t \), and \( o' \) does not have a report that is created after 0. The probability for this to happen is \( q(t) \cdot (1 - \frac{\mu}{N} \cdot \Delta t) \cdot \lambda \cdot \Delta t \cdot (1 - g(t)) \).

3. At time \( t \) \( o \) has no reports, or has a report that is created before 0, and it does not acquire any new report between \( t \) and \( t+\Delta t \), and it interacts with one peer \( m' \) between \( t \) and \( t+\Delta t \), and \( m' \) has \( R \). The probability for this to happen is \( (1 - q(t) - g(t)) \cdot (1 - \frac{\mu}{N} \cdot \Delta t) \cdot \lambda \cdot \Delta t \cdot q(t) \).

Thus

\[
q(t + \Delta t) = q(t) \cdot (1 - \lambda \cdot \Delta t) \cdot (1 - \frac{\mu}{N} \cdot \Delta t) + q(t) \cdot (1 - \frac{\mu}{N} \cdot \Delta t) \cdot \lambda \cdot \Delta t \cdot (1 - g(t)) +
\]

\[
(1 - q(t) - g(t)) \cdot (1 - \frac{\mu}{N} \cdot \Delta t) \cdot \lambda \cdot \Delta t \cdot q(t)
\]

By similar analysis, we obtain the following equation:

\[
g(t + \Delta t) = \frac{\mu}{N} \cdot \Delta t + (1 - \frac{\mu}{N} \cdot \Delta t) \cdot g(t) + (1 - \frac{\mu}{N} \cdot \Delta t) \cdot (1 - g(t)) \cdot \lambda \cdot \Delta t \cdot g(t)
\]

After simplification of the above difference equations, we get the following differential equations:

\[
\begin{cases}
\frac{dq(t)}{dt} = -(\lambda + \frac{\mu}{N}) \cdot q(t) + 2 \cdot \lambda \cdot q(t) \cdot (1 - g(t)) - \lambda \cdot q(t)^2 \\
\frac{dg(t)}{dt} = \frac{\mu}{N} \cdot g(t) + \lambda \cdot g(t) - \lambda \cdot g(t)^2
\end{cases}
\]

(4.5.1)

Since each peer is equally probable to acquire the report, \( q(0) = 1/N \). Finally, \( g(0) = 0 \). Let \( C(t) \) be the number of copies of a report \( t \) time units after its creation. We have the following theorem.
**Theorem 4.4:** $C(t)$ is a random variable with expected value $q(t) \cdot N$ where $q(t)$ is given by the equation group (4.5.1).

We used Theorem 4.4 to compute the expected number of copies as a function of time. We used the following set of parameter values: $N=2500$, $\lambda=0.12$, $\mu=10$. The solid line in Figure 40 shows the result. Observe that the number first increases until a maximum value is reached. And then it decreases until disappearing from the system. From this figure we can estimate how far away a report can be propagated. Take the cut-off age beyond which the expected number of copies is below 1, which is about 60 seconds. Assume that the wireless transmission range is zero. Multiplying this cut-off age by the maximum speed of moving objects gives the maximum distance the report can be propagated to. For example, if the maximum speed is 60 miles/hour, then the maximum distance is 1 mile.

4.5.1.3 Validation of the Mathematical Model

We conducted a simulation to validate the analytical model. In this simulation, 2500 objects are initially uniformly distributed within a 5mile×5mile square area and they randomly move with a constant speed 40 miles/hour. The transmission range is 50 meters. This setup gives on average 0.12 interactions per each object per each second. Reports are generated with intensity 10 and each report is randomly assigned to an object. Figure 40 shows the results. The dashed line represents the experimental result. It can be seen that Theorem 4.4 accurately describes the behavior of the system.
4.5.2 Propagation Pattern of Reports for Spatio-temporal Resources

In this subsection we study by simulations the propagation pattern of reports for spatio-temporal resources. First we describe the simulation method and then we present the simulation results.
4.5.2.1 Simulation Method

We synthetically generated and moved objects within a 10mile×10mile square area. The objects move in a random-walk model. Specifically, for each object $i$, we randomly chose two points within the square area, and assigned them as the start point and the first stop of $i$ respectively. $i$ moves along line segment between the two points at a constant speed. When the first stop is reached, another random point is chosen as the second stop of $i$, and $i$ moves from the first stop to the second stop at the same constant speed. And so on. The motion speed of $i$ is randomly picked up from the interval $[v-5, v+5]$ where $v$ is a parameter.

Hotspots are randomly distributed in the square area with density 500 hotspots per square mile. Resources are generated only at hotspots. At each hotspot, the length of the valid duration of a resource follows an exponential distribution with mean $u$ seconds, and the time length of the invalid duration follows an exponential distribution with mean 360 seconds. The home of all the reports announced by a hotspot is the location of the hotspot. All the hotspots and the moving objects have the same wireless transmission range. All the moving objects have the same interest threshold. The value of the time decay factor $\alpha$ is $1/u$ and the distance decay factor $\beta$ is $1/(u \cdot s)$ where $s$ is the motion speed of a moving object.

There are five parameters for each simulation run, namely the interest threshold $M$, the wireless transmission range $r$, the constant speed $v$, the objects density $g$ (i.e. the number of objects per square mile), and the mean of valid duration $u$. $M$ is fixed to be 10, $v$ is fixed to be 40 miles/hour, and $u$ is fixed to be 120 seconds. The time unit is second. All the parameters and their values are listed in Table X.
Table X: Simulation parameters and their values.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Unit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean of valid duration</td>
<td>$u$</td>
<td>second</td>
<td>120</td>
</tr>
<tr>
<td>Interest threshold</td>
<td>$M$</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>Transmission range</td>
<td>$r$</td>
<td>meter</td>
<td>50</td>
</tr>
<tr>
<td>Motion speed</td>
<td>$v$</td>
<td>miles/hour</td>
<td>40</td>
</tr>
<tr>
<td>Object density</td>
<td>$g$</td>
<td>objects/mile$^2$</td>
<td>100</td>
</tr>
</tbody>
</table>

Each simulation run is executed as follows. At the beginning of the simulation run, 10$\times$10$\times g$ objects are generated and they start to move at the same time (time 0). Resources are generated and the status of each resource alternates between valid and invalid as described earlier. When the distance between two peers is smaller than $r$ in a time unit, they exchange their reports, re-evaluate the relevance, and purge the least relevant reports if needed. Each exchange is finished instantaneously. The length of each run is 10 simulated hours.

During a simulation run, we trace the distribution of each report $a(R)$ at each time unit. For this purpose, we generate 50 rings centered at the home of $R$, each with the width of 0.05 mile. At each time unit we calculate the density of the copies of $a(R)$ at each ring, and average among all the time units of a simulation run.

### 4.5.2.2 Simulation Results

Figure 41 shows the average density as a function of the age and the distance to home for MARKET. The density is coded by the gray-level, such that a deeper gray-level represents a higher density. The lowest gray-level (white) represents zero density. We make the following observations.

**Observation 1:** At any point in time, there is a spatial boundary for the distribution of the report, beyond which the density is zero. This boundary first expands, until a maximum value (about 0.9 mile) is reached. Then the boundary starts to shrink until finally the report disappears from the system (at the age
of 450 seconds or so). The boundary expands at beginning because of the propagation of the report caused by opportunistic exchanges. However, as time passes, the relevance decreases, causing two effects: (i) more objects purge the report out; and (ii) less objects save it. These two effects make the number of copies start to decrease, and thus the boundary starts to shrink. After some time, the relevance becomes so low that all the objects that have carried it have purged it out, and no objects save it upon exchange. The report thus disappears from the system.

**Observation 2:** The gray-level tends to be deep when the distance to the home is small and it fades as the distance to home increases. In other words, the copies are more densely distributed in the areas close to the home than in the areas farther away. This is a useful behavior, because it means that the report has a higher availability in the area to which it is of interest.

The above propagation pattern shows that, by very simple local decisions made at each moving object, the opportunistic dissemination algorithm automatically limits the global distribution of a report to a bounded spatial area, which is a circle around the home location of the resource. The algorithm also limits the distribution to the time-duration for which the report is of interest. We conducted experiments with more parameter configurations. These experiments show that the spatial and temporal boundaries automatically adapt depending on the number of resources in the system, the traffic density and speed, and other parameters that dictate the amount of storage, processing power, and bandwidth that should be allocated to each resource. For example, if resources are generated less frequently, then each report will stay in the system longer, and spread farther.

### 4.5.3 Relevant Work

The propagation of reports in mobile P2P networks has been analyzed in [66] and [67] using the epidemic model [68]. The propagation in these papers concerns the propagation of a single report. The competition of multiple reports on memory consumption and bandwidth consumption is not taken into account. In our case, we investigated the propagation of multiple reports with the memory and bandwidth constraints taken into account. This characteristic in our case is expressed by the relevance function.
Because the relevance of a report for a mobile peer is dynamic, a mobile peer may refuse a report at one time and then later find it of interest.
5 CONTINUOUS NEAREST-NEIGHBOR QUERIES WITH LOCATION UNCERTAINTY

A $Ck$NN query is a query that continuously finds the $k$ nearest mobile peers with respect to a given mobile peer. Processing of $Ck$NN queries can be enabled by mobile peers disseminating their locations and velocity-vectors via P2P communication. In this way each mobile peer maintains a database that stores the motion information (locations and velocity-vectors) of each other mobile peer. Due to GPS errors and delay/disruption of the mobile P2P network, there can be uncertainty associated with mobile peer locations. In this chapter we discuss $Ck$NN queries with location uncertainty. The difference between this chapter and section 4.4 of Chapter 4 is that the latter considers instantaneous rather than continuous $k$NN queries. That is, in section 4.4 of Chapter 4, a $k$NN query only asks for the $k$-nearest neighbors that pertain to the time at which the query is issued. On the other hand, in this chapter a $k$NN query asks for the $k$-nearest-neighbors throughout the time; it requires that the answer be updated whenever the set of $k$-nearest-neighbors changes.

In section 5.1 we summarize contributions of our work and discuss relevant work. At the end of this section we outline the structure of this chapter. Throughout this chapter we use the terms mobile peers and moving objects interchangeably.

5.1 Contributions and Relevant Work

5.1.1 Contributions

A $Ck$NN query is a query that continuously finds the $k$ nearest neighbors with respect to a given moving point-object $O_q$ among a set of $n$ moving point-objects. The query returns a sequence of answer-pairs, namely pairs of the form $(I, S)$ such that $I$ is a time interval and $S$ is the set of $k$ objects that are nearest to $O_q$ during $I$. $Ck$NN queries see many applications in mobile computing environments. For
example, in a road network, the CkNN query can continuously provide a driver with the locations of the k nearest vehicles from her location. The query enables the driver to be aware of the vehicles that are blocked by a truck, a turn, a blind zone, etc. In a digital battlefield, a vehicle may use the CkNN query to monitor the k nearest hostile (or friendly) vehicles for surveillance (or for support). CkNN queries are also used for clustering data streams ([84]). In this case moving objects are sensor observations rather than physical objects and they do not necessarily move in a geo-space. In general, CkNN queries can be used in any database application that needs to continuously rank answers based on their Euclidean distances to an object.

Now we discuss various models of the CkNN query.

**Motion Model:** We discuss two motion models, linear and piece-wise linear. In the linear model all the objects, including $O_q$, start from different points and move with a constant velocity-vector in a multi-dimensional space. This model is applicable when the database only knows the initial location and velocity-vector of each moving object. In the piecewise linear model the objects change their velocities a finite number of times denoted by $m$. Here are some scenarios in which the piecewise linear model is useful. **Scenario 1:** The database is given a sequence of historical records for each moving object where each record indicates that the object was at a particular location at a particular time. If we assume that the object moves linearly between two consecutive records, then the motion of the object follows the piecewise linear model. **Scenario 2:** Each object sends updates to the database from time to time, where each update includes the object’s current location and velocity-vector. The database assumes that the object will move linearly using the current velocity-vector until the next update is received. In this case the motion of the object follows the piecewise linear model as far as the database is concerned. **Scenario 3:** The database knows the motion plans of each object such as a delivery truck. A motion plan describes the object’s future path in a road network and a schedule. For example, the object is supposed to arrive at location $p_i$ at time $t_i$ and so on. Based on the motion plan, the future motion of the object can be described by the piecewise linear model. **Scenario 4:** The piecewise linear model can approximate a non-linear motion with an arbitrarily high precision.
Certainty model: The certain C$k$NN query assumes that the location of an object as a function of time is known precisely. In this case, for each answer-pair $(I, S)$, $S$ is exactly the set of $k$ objects that are nearest to $O_q$ during time interval $I$. However, uncertainty is an inherent aspect in databases that manage locations of moving objects. Due to measurement imprecision (such as GPS errors), continuous motion, and network delays, the location of a moving object stored in the database will not always precisely represent its real location. In some scenarios, location uncertainty is deliberately introduced. For example, in order to protect privacy in location based services, uncertainty is injected to a user’s location (see e.g., [85]). Specifically, we consider the uncertainty model in which at any point in time the location of a moving object can be anywhere in a circular region, but not outside it (see e.g., [86, 87]). Thus, due to the uncertainty of object locations, the set of actual $k$ nearest neighbors is unknowable. In other words, at any point in time there may be more than $k$ objects that are possibly the $k$ nearest neighbors. We define a CP$k$NN (continuous possible $k$NN) query which is to continuously find all the objects that are possibly the $k$ nearest neighbors. The query returns a sequence of answer-pairs, namely pairs of the form $(I, S)$ such that $I$ is a time interval and $S$ is a set of objects. $S$ may contain more than $k$ objects, but each member $x$ of $S$ may be a $k$ nearest neighbor. In other words, for each object $o$ there is a location $lo$ within $o$’s circular uncertainty region, such that if each $o$ is at $lo$, then $x$ is a $k$ nearest neighbor. And conversely, if $x$ is not in $S$, then regardless where the objects are within their uncertainty regions, $x$ cannot be a $k$ nearest neighbor. In this sense, even though the data is uncertain, the answer to a CP$k$NN query is precise.

Processing model: We consider two different processing models, namely off-line processing and on-line processing. Off-line processing assumes that the complete object-trajectories are given a priori, and it computes all the answers. A database update is insertion of a new object, deletion of an existing object, or velocity-vector change of an existing object. For each update the off-line processing re-computes all the answer-pairs. In contrast, on-line processing assumes that trajectories are given dynamically and incrementally. That is, the processing algorithm only knows the trajectory of each object up to the present point in time. Each update extends a trajectory, starts a new trajectory, or ends a trajectory. The
processing algorithm returns the current answer whenever it changes. When a trajectory-update is received, the on-line processing only re-computes the current answer.

In §5.1.1.1 we discuss, for the various dimensions of the problem, the value of $\beta_k(n)$ which represents the maximum number of pairs in an answer to the C$k$NN problem with $n$ moving objects. In §5.1.1.2 we discuss the algorithms for the various dimensions of the problem. The analysis of these algorithms uses $\beta_k(n)$.

### 5.1.1.1 Number of answer pairs

In the first part of this chapter we study the value of $\beta_k(n)$ which represents the maximum number of pairs in an answer to the C$k$NN problem with $n$ moving objects. This enables complexity analysis for C$k$NN query processing in the second part of the chapter. Throughout the chapter we assume that $k$ is a constant. We study both the lower bound and the upper bound of $\beta_k(n)$.

We start with the linear motion model. For this model, the tightest upper bound that is known up to date is $8k(n-k)+1$ (see Theorem 3.1 in [88]). In this chapter we give an exact upper bound of $k(2n-k+1)+1$. Since $k(2n-k+1)<8k(n-k)$ when $k\leq 6n/7$, which holds for most real applications, our bound is tighter.

Now the question is whether the $O(n)$ upper bound is attainable. In other words, what is the lower bound of $\beta_k(n)$. In this chapter we show that $O(n)$ is also the lower bound of $\beta_k(n)$. We show this by constructing a feasible configuration in terms of the motion of objects such that $\beta_k(n)$ is equal to $2(n-k)+1$.

Then we bound $\beta_k(n)$ for the piecewise linear model. For this model the tightest known upper bound for $\beta_k(n)$ is $O(nm^2\alpha(nm))$ (see Theorem 2.4 in [89]) where $\alpha$ is the functional inverse of Ackermann’s function (see [89] for the definition of Ackermann’s function). In this chapter we prove a tighter bound of $O(nm\alpha(n))$. We further show that the lower bound of $\beta_k(n)$ for the piecewise linear model is $2m(n-k)+1=O(nm)$. Given the fact that $\alpha(n)$ is at most 4 for any practical value of $n$, the bound of $O(nm\alpha(n))$ is nearly tight as it only adds an almost-constant factor to the lower bound.
Then we examine CP\(k\)NN, i.e. C\(k\)NN under the condition that there is uncertainty associated with the locations of moving objects. Our analysis shows that the bound on \(\beta_k(n)\) in the uncertain case is \(O(n)\) times the corresponding bound for the certain case. For example, the bound for the linear model is \(\Theta(n^2)\) in comparison with \(\Theta(n)\) in the certain case; the bounds for the piecewise linear model are \(O(n^2mt\alpha(n))\) and \(\Omega(n^2m)\) in comparison with \(O(nmt\alpha(n))\) and \(\Omega(nm)\) in the certain case.

5.1.1.2 C\(k\)NN algorithms

In the second part of the chapter, we use the results of the first part to study the processing of C\(k\)NN queries in the certain and uncertain cases, respectively. For off-line processing in the certain case, there is an existing simple divide-and-conquer algorithm which gives the solution for the linear model in \(O(n\log n)\) time (see Theorem 2.6 in [89]). For on-line processing in the certain case, we develop a kinetic data structure, called object heap. This data structure allows updates like insertion of a new object, deletion of an existing object, and velocity-vector change of an existing object. We analyze the complexity of object heap under two different conditions, depending on whether there are updates or not. In either case, we assume that initially each object moves linearly until updated. When there are no updates, the cumulative complexity of object heap, i.e., the total cost for returning all answer-pairs (Recall that on-line processing returns one answer-pair each time), is \(O(n\log^2 n)\). This is the same as that of kinetic tournament [91] and kinetic heap [91, 92], two classical kinetic data structures for monitoring the nearest neighbor (i.e., \(k=1\))\(^9\). However, object heap is better at handling updates. In object heap, insertion, deletion, and velocity-vector change can be carried out in \(O(\log n)\) time each. In kinetic tournament and kinetic heap, these operations require \(O(\log^2 n)\) time each. We prove that if each object can change its velocity-vector at most \(m\) times, then the cumulative complexity of our on-line algorithm is \(O(nmt\alpha(n)\log^2 n)\) which is higher than the lower bound by a factor of \(\alpha(n)\log^2 n\). The theoretical results of this chapter are summarized in Table XI.

\(^9\) Better complexity is known for kinetic heap when the distance functions are pseudo-lines (i.e., any pair of distance functions intersect each other at most once) (see [7]). In our problem, the square-distance functions are parabolas. Observe that each parabola can be cut into two pseudo-lines. However, this will make the square-distance functions partially defined whereas the analysis in [7] assumes totally defined functions.
Then we compare experimentally different algorithms using a database of 1 million objects derived from real-world GPS traces. The results are shown in Figures 55 through Figure 69 in §5.6. These results show that even though object heap is better than kinetic tournament by a factor of $\log(n)$ in the worst case, it is equally efficient as the latter one for the data set we tested, in the average case. The results also show that on-line processing is more efficient than off-line processing for practical usage. The experimental results also show that the number of answer-pairs grows almost linearly with number of objects for the piecewise-linear certain case. They also show how the number of answer-pairs grows in the uncertain case.

In summary, the main contributions of this chapter are the following:

1. We bound the number of answer-pairs in a $C_k$NN query for various scenarios in terms of the motion model and whether there is uncertainty associated with locations. Some of the bounds that we provide have not been studied before. The others are tighter than existing bounds.

2. We introduce an algorithm called object heap for on-line processing of $C_k$NN queries. The worst case complexity of object heap is lower than that of existing on-line algorithms by a factor of $\log(n)$.

3. We develop an uncertain version of object heap.

4. We evaluate the scalability of the proposed algorithms by experiments using real-world data.
Table XI. Summary of the results.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Our Result</th>
<th>Previous Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Certain Case</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of Answer-Pairs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linear</td>
<td>lower</td>
<td>$2(n-k)+1$</td>
</tr>
<tr>
<td></td>
<td>upper</td>
<td>$k(2n-k-1)+1$</td>
</tr>
<tr>
<td>Piecewise Linear</td>
<td>lower</td>
<td>$2m(n-k)+1$</td>
</tr>
<tr>
<td></td>
<td>upper</td>
<td>$O(nma(n))$</td>
</tr>
<tr>
<td>Off-line Processing, Piecewise Linear</td>
<td>$O(nma(n)\log n)$</td>
<td>$O(nm\log(nm))$*</td>
</tr>
<tr>
<td>On-line Processing</td>
<td>Single Update</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td></td>
<td>Cumulative Complexity with $m$ Velocity-Vector-Changes per Object</td>
<td>$O(nma(n)\log^2 n)$</td>
</tr>
<tr>
<td><strong>Uncertain Case</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of Answer-Pairs</td>
<td>Linear</td>
<td>$\Theta(n^2)$</td>
</tr>
<tr>
<td></td>
<td>Piecewise Linear</td>
<td>$O(n^2ma(n)), \Omega(n^2m)$</td>
</tr>
<tr>
<td>Off-line Processing, Piecewise Linear</td>
<td>$O(n^2ma(n)\log n)$</td>
<td></td>
</tr>
<tr>
<td>On-line Processing, Cumulative Complexity with $m$ Velocity-Vector-Changes per Object</td>
<td>$O(n^2ma(n)\log n)$</td>
<td>not studied**</td>
</tr>
</tbody>
</table>

* Results obtained by a straightforward application of existing techniques.

** For off-line processing in the piecewise linear model in the uncertain case, [93] gives an upper bound of $O(n'm)$ for a more complex query semantics.

5.1.1.3 Comparison with other Relevant Work

Relationship with other indexing work

We discuss each result listed in Table XI in terms of its applicability in disk storage databases and its combination with indexing structures. First we consider the certain case. All the results for the number of answer-pairs are applicable to both disk storage databases and main memory databases, regardless of whether indexing structures are used or not. For off-line processing, there are two cases depending on whether the database is disk based or main-memory based. In the case of disk storage databases, query processing typically requires a combination of filtering and refinement stages ([94, 95, 96, 97]). In the filtering stage, indexing structures are used to prune the objects that are not candidates to be in the answer set. In the refinement stage the candidate objects are examined in main memory, and our off-line processing algorithm applies to the refinement stage. In the case of main memory databases, our algorithm is applicable verbatim. Now consider on-line processing. Our on-line processing algorithm (i.e., object heap) is not directly applicable to disk storage databases as is. In the case of main memory databases, the object heap provides indexing with worst-case guarantees, whereas spatial indexing methods such as R-tree do not.
The above discussion applies verbatim to the uncertain case as well.

**CkNN in the certain case.**

The processing of CkNN queries has been extensively studied in the database community (see e.g., [90, 98, 99, 100, 101]). However, existing studies lack complexity analysis due to a limited understanding of the maximum number of answer-pairs. These studies either do not provide algorithm complexity analysis, or treat the number of answer-pairs as an input size that is independent of $n$. In this chapter we provide a thorough analysis of the maximum number of answer-pairs, for both the certain case and the uncertain case. Our results enable complexity analysis of CkNN algorithms, as demonstrated in this chapter.

Li et al. [98] introduce a CkNN algorithm called Beach-line. The algorithm monitors the $k$-th NN (so called beach-line) since the necessary condition of changes in the $k$NN set is the change of the $k$-th NN for CkNN. The paper does not provide complexity analysis. Iwerks et al. [90] introduce a C1NN algorithm, the idea of which is similar to the Beach-line algorithm. In that work, the authors present a complexity bound of their algorithm in terms of the number of answer-pairs as an additional parameter. Using our results on the number of answer-pairs (Theorem 5.1), it can be shown that the complexity of the Beach-line algorithm is $O(n^2)$ in the linear model.

Mokhtar et al. [102] proposes a general method of processing moving object queries including CkNN queries. They assume a piecewise linear motion model. They have a general result that a moving object query such as a CkNN query can be processed in polynomial time in terms of the database size. In this chapter we provide a more specific result that a CkNN query can be processed in $O(nm\alpha(n) \log n)$ time in the piecewise linear model.

CkNN queries are studied in [96] in the context where only the query object (i.e., the object that the query pertains to) moves. Since other objects (i.e., data objects) are stationary, they are stored in an R*-tree. Their approach is to use range search to find $k$ closest objects and re-calculate the range at each update on the query object using the moving distance since the last update. This gives a correct query result only at the time of search following the update, and the result may soon become incorrect due to the
movement of the query object. In our case, both the query object and the data objects can be moving, and the algorithms analyzed in this chapter guarantee the correctness of answers.

Kolahdouzan et al. [100] study CkNN queries in spatial network databases, where the distance between two objects is a function of the network connectivity (e.g., shortest path between two objects). In this chapter we consider Euclidean distances. Furthermore, in their case the data objects are static and thus they are able to utilize the Voronoi diagrams to partition the network and facilitate query processing. In our case, the data objects can move. Intuitively constructing and maintaining Voronoi diagrams in this case is inefficient.

Xiong et al. [101] propose an on-line algorithm for CkNN queries where both the query object and the data objects can move. In their model, the actually positions but not velocities of the objects are periodically updated. From the actual positions the kNN set is computed using an incremental algorithm. On the other hand, we assume that initial locations and velocities are transmitted only once in the linear model, using which the kNN set is computed for all the future time points. Of course we also allow updates to velocities in the piecewise linear model.

**CkNN in the uncertain case.**

Several studies have been done concerning CkNN with location uncertainty. Trajcevski et al [86, 93] deal with k=1 only and they consider a different query semantics. In their case, a C1NN query asks continuously for a ranking of objects in the order of their probabilities to be a NN object. They give an upper bound of $O(n^2)$ on the number of answer-pairs in their query semantics for linear motion. We show that this bound is the same in the simpler semantics studied in this chapter and it is tight. For piecewise linear motion they give an upper bound of $O(n^3m)$ whereas in our case it is $O(n^2m\alpha(n))$. Furthermore, [86, 93] do not deal with $k>1$ and on-line processing, whereas we do so in this chapter.

Huang et al. [94, 103, 104] assume a different uncertainty model, in which the velocity-vector of each moving object follows a probability distribution that is known to query processing. Our uncertainty model does not require the knowledge of the velocity-vector distribution. Furthermore, they do not provide any complexity analysis and they do not deal with on-line processing and piecewise linear model.
Probabilistic queries.

Probabilistic queries are queries that evaluate data uncertainty and provide probabilistic guarantees ([84, 87, 105, 106, 107, 108]). The answer to a probabilistic $k$NN query lists not only the objects that are possibly the nearest neighbors but also the probability for each of them to be the nearest neighbor. Probabilistic queries are more informative than “qualitative” queries studied in this chapter. The probabilistic version of a CP$k$NN query is: “Continuously find all the objects that are possibly the $k$ nearest neighbors along with probabilities”. However, due to the difficulty of query processing, existing studies are limited to instantaneous queries. Cheng et al. [87, 106, 108] propose efficient computation and indexing algorithms for evaluating probabilistic $k$NN queries. However, their algorithms only deal with instantaneous rather than continuous queries. In other words, the answer returned by their algorithms only pertains to the locations of the moving objects at the time when the answer is evaluated. The answer does not predict for how long the probability values will be valid. Chen et al. [84] propose algorithms for processing “continuous” probabilistic NN queries. These queries are said to be continuous in the sense that they reside in the database for an extensive amount of time. For these queries, the answer is recomputed whenever there is a location update to the database. The paper develops an incremental evaluation technique so that each update may reuse the query results computed for the previous update. But again, each answer only pertains to the locations at the time when the answer is evaluated. To the best of our knowledge, none of the existing algorithms is able to process the probabilistic version of the CP$k$NN query.

The rest of this chapter is structured as follows: §5.5.2 analyzes the bound on the number of answer-pairs for the certain case. §5.5.3 analyzes the bound for the uncertain case. §5.5.4 discusses query processing in the certain case. §5.5.5 discusses query processing in the uncertain case. §5.5.6 concludes the chapter and discusses some open issues that are worth further study.
5.2 **Continuous $k$NN with Certain Location**

In this section we analyze the maximum number of answer-pairs for a $Ck$NN query in the certain case. We do this analysis for two reasons. First, the results obtained from this analysis will be used in the analysis for the uncertain case. Second, the results for the uncertain case will become interesting when compared with the results for the certain case.

The rest of this section is organized as follows. In §5.2.1 we discuss the number of answer-pairs in the case where objects move linearly. In §5.2.2 we discuss the number of answer-pairs in the case where objects move piecewise linearly.

5.2.1 **Number of Answer-pairs with Linear Motion**

We consider query processing at a query object $O_q$ that is moving. $O_q$ has an objects database that stores the motion information of each other data object $O_1, O_2, ..., O_n$ that is also moving. The $Ck$NN query requests for every point in time the $k$ nearest neighbors of $O_q$ among all the data objects. All the objects move in a linear manner in a multi-dimensional space called the motion space. That is, each object moves along a straight line with a constant speed. The motion space can have an arbitrarily high number of dimensions. In other words, our results apply to an arbitrary number of dimensions. All the objects start at negative infinity and continue to positive infinity. Then the square of the distance $d_i$ between $O_q$ and a data object $O_i$ is a quadratic function of time $t$: $d_i^2(t) = at^2 + bt + c$, where $a$, $b$, and $c$ are parameters dependent on the velocities and initial locations of $O_q$ and $O_i$. The coefficient $a$ is non-negative and thus the $d_i^2(t)$ function is convex. The $d_i^2(t)$ function is a parabola in the Time-SquareDistance space [98] as illustrated in Figure 42. For convenience of presentation, in the rest of this chapter we use object $O_i$ and its square-distance function $d_i^2(t)$ interchangeably when there is no confusion.
The CkNN query is issued at time 0. At that time $O_q$ has a set of $k$ nearest neighbors, but as time progresses, the $k$NN set may change. A time interval during which the $k$NN set remains unchanged is referred to as an answer interval. An answer-pair is a pair consisting of an answer interval and its associated $k$NN set. Answer-pairs are required to be non-redundant in that, for any pairs $(I, S)$ and $(I', S')$, the sets $S$ and $S'$ are distinct if $I$ and $I'$ are contiguous. As an example, in Figure 42, if $k=2$, then the answer-pairs are:

$$<[0, t_1), \{O_1, O_2\}>,$$

$$<[t_1, t_2), \{O_2, O_3\}>,$$

$$<[t_2, t_3), \{O_1, O_3\}>,$$

$$<[t_3, \infty), \{O_1, O_2\}>$$

Denote by $I_k(G)$ the number of answer-pairs for an instance $G$ of the objects database for the CkNN query. Define

$$\beta_k(n) = \max \{I_k(G) | G \text{ is an instance of the objects database with } n \text{ objects}\}$$

In words, $\beta_k(n)$ is the maximum number of answer-pairs for the CkNN query among all the objects database instances with $n$ objects. First we discuss the upper bound of $\beta_k(n)$. The following definition is used in the discussion.
Definition 2.1. Let \( p \) be an intersection of two objects \( A \) and \( B \). We use \( p.time \) to denote the time coordinate of the intersection. \( p \) is associated with another two attributes, namely its downward and upward. \( A \) is the **downward** of \( p \), and \( A \) **downward-crosses** \( B \) at \( p \), if \( A \) is farther than \( B \) immediately before \( p.time \) (and thus closer than \( B \) immediately after \( p.time \)). In this case \( B \) is the **upward** of \( p \), and \( B \) **upward-crosses** \( A \) at \( p \).

Clearly, for any two objects \( A \) and \( B \), \( A \) downward-crosses \( B \) at most once and \( A \) upward-crosses \( B \) at most once.

Now we examine the upper bound of \( \beta_k(n) \). Observe that for any objects database instance, the number of answer-pairs is upper bounded by the number of times the \( k \)-th nearest neighbor changes. This is because each change of the answer set is always caused by a change of the \( k \)-th nearest neighbor whereas a change of the \( k \)-th nearest neighbor does not necessarily cause a change of the answer set. Specifically, the answer set changes only when there is a switch of order between the \( k \)-th nearest neighbor and the \((k+1)\)st nearest neighbor. In the Time-SquareDistance space, such a switch occurs when a \( k \)-th lowest curve upward-crosses a curve. For example, in Figure 42, at point \( p_1 \), the 2nd lowest curve \( O_1 \) is crossed by \( O_3 \) from above, and thus the answer set is changed at \( p_1 \). Specifically, \( O_1 \) is removed from the answer set, and \( O_3 \) is added to the answer set. On the other hand, the answer set does not change when a \( k \)-th lowest curve downward-crosses a curve. For example, in Figure 42, at point \( p_0 \), the 2nd lowest curve \( O_2 \) is crossed by \( O_1 \) from below, and thus the answer set is not changed at \( p_0 \).

According to the above observation, the number of answer-pairs is upper bounded by the number of pieces in the envelope formed by the \( k \)-th lowest curves. The latter number has been studied in the context of arrangements of curves and is formally defined as follows (see [89]). Given a set \( \Gamma \) of curves where each curve is a continuous and univariate function, the arrangement \( A(\Gamma) \) of \( \Gamma \) is the planar subdivision induced by the curves in \( \Gamma \). That is, \( A(\Gamma) \) is a planar map the **vertices** of which are the pair-wise intersection points of the curves in \( \Gamma \) and the **edges** of which are maximal connected portions of the curves that do not contain a vertex. As an example, Figure 42 shows the arrangement of three objects \( O_1 \),
$O_2$, and $O_2$, where white circles are vertices. The level of a point $p$ in $A(I)$ is the number of curves in $\Gamma$ not above $p$, and the level of an edge $e \in A(I)$ is the common level of all the points lying in the relative interior of $e$\textsuperscript{10}. The $k$-level of $A(I)$ is the union of all edges in $A(I)$ whose level is $k-1$. The length of the $k$-level is the number of edges in the $k$-level. In Figure 42, the thick curve shows the 2-level in the arrangement of $O_1$, $O_2$, and $O_3$; its length is 7.

It is well known that the length of the $k$-level in an arrangement of $n$ parabolas, each with an axis of symmetry that is parallel to the $y$-axis, is $O(n)$ for any constant $k$ (see e.g. [109]). However, most of the existing literature only gives order-statistics results. Theorem 3.1 in [88] implies an exact bound of $8k(n-k)+1$. In the following theorem we give a tighter exact bound.

**Theorem 5.1.** Assume that the query object and data objects all move linearly. Then for any constant $k \beta_k(n) \leq k(2n-k-1)+1$.

*For the proof of Theorem 5.1 we give the following definition. Given the arrangement of data objects in the Time-SquareDistance space, an intersection point in the arrangement is referred to as a $\leq k$-level intersection (respectively $>k$-level intersection) if it is an endpoint of an edge the level of which is smaller than or equal to $k$ (respectively greater than $k$).

**Proof of Theorem 5.1.** We prove by showing that the number of $\leq k$-level intersections is at most $k(2n-k-1)$. Without loss of generality, we assume that at the time when the query is processed, $O_1$ is the first nearest neighbor, $O_2$ is the second nearest neighbor, ..., and $O_n$ is the $n$-th nearest (and the farthest) neighbor (see Figure 43(a)). We construct the arrangement in the same order. That is, we add $O_1$ first, and then add $O_2$, and so on. We show that for any integer $k \leq j \leq n$, the addition of $O_j$ can introduce at most $2k$ $\leq k$-level intersections to the final arrangement of the $n$ objects. To do this, we divide the intersections introduced by $O_j$ to the arrangement of the first $j$ objects into two groups, namely the $\leq k$-level

\[\text{In the computational geometry literature, the level of a point } p \text{ is defined to be the number of curves lying strictly below } p. \text{ In this paper we define the level to be the number of curves not above } p, \text{ so that the } k\text{-level corresponds to the } k\text{-th nearest neighbor. But this definition is purely terminological without affecting any results.}\]
intersections and \( >k \)-level intersections. We call these two groups *lower intersections* and *higher intersections* respectively. Observe that, due to the later introduction of \( O_{j+1}, O_{j+2}, \) and so on, some of the lower intersections may become \( >k \)-level intersections in the final arrangement. However, none of the higher intersections may become \( \leq k \)-level intersections in the final arrangement. Thus the number of \( \leq k \)-level intersections introduced by \( O_j \) to the final arrangement is upper bounded by the number of lower intersections. In the following we show that the number of lower intersections is at most \( 2k \).

If \( O_i \) never reaches the \( k \)-th position, then the statement clearly holds. If \( O_j \) does reach the \( k \)-th position, then consider the first time it reaches the \((k+1)\)st position. Denote that time by \( T \). Observe that \( O_j \) has to experience a series of downward-crosses to reach the \((k+1)\)st position. We refer to the objects that are closer to \( O_q \) than \( O_j \) at time \( T \) as the *lower objects* and the objects that are farther away from \( O_q \) at time \( T \) as the *higher objects*. Clearly there are \( k \) lower objects. In Figure 43(b), the objects below \( O_j \) are the lower objects. The objects above \( O_j \), including \( O_{j+1}, O_{j+2}, \ldots, O_n \), are the higher objects. Notice that the set of lower objects and that of higher objects pertain to time \( T \).

Assume that object \( O_j \) participates in \( d \) lower intersections with the higher objects after time \( T \). Obviously, \( O_j \) can make at most \( 2k \) lower intersections with the lower objects. Observe that for each lower intersection that \( O_j \) makes with an higher object after time \( T \), that higher object will move to the left of \( O_j \) and it can never move to the right of \( O_j \) again, because the downward-cross has already been consumed before time \( T \).

Now observe that if more than \( k-d \) of the lower objects make \( 2 \) lower intersections with \( O_j \), it means that an higher object upward crosses \( O_j \) after time \( T \); thus this is impossible. Therefore at most \( k-d \) of the lower objects make \( 2 \) lower intersections with \( O_j \) and each one of the rest makes at most one lower intersection with \( O_j \). Thus if the number of lower intersections of higher objects with \( O_j \) is \( d \), then the maximum number of lower intersections of lower objects with \( O_j \) is \( 2k-d \). Thus the maximum number of lower intersections is \( 2k \).
Figure 43. The auxiliary figure for the proof of Theorem 5.1.

So far we have studied the case of $k < j \leq n$. Using similar argument we can show that in the case of $j \leq k$ the number of $\leq k$-level intersections introduced by $O_j$ to the final arrangement is $2(j-1)$.

In summary, the total number of $\leq k$-level intersections in the final arrangement is:

$$2k(n-k) + \sum_{j=1}^{k} 2(j-1) = k(2n-k-1)$$

The length of the $k$-level is upper bounded by the number of $\leq k$-level intersections plus one. Thus the number of answer-pairs is at most $k(2n-k-1) + 1$. $\square$
For some $k$’s, the exact number given by Theorem 5.1 is tight. For example, when $k=1$, the number of answer-pairs is equal to the length of the 1-level which is tightly bounded by $2n-1$ according to Theorems 2.1 and 3.1 in [89]. Theorem 5.1 gives the same bound in this case. The order given by Theorem 5.1 (i.e., $O(n)$) is tight for all values of $k$, as shown by the following proposition.

**Proposition 5.1.** Assume that the query object and data objects all move linearly. Then for any constant $k$, $\beta_k(n) \geq 2(n-k)+1$.

**Proof:** We construct a feasible case in which the number of answer-pairs is linear in $n$. Let objects $O_q$, $O_1$, $O_2$, ..., and $O_k$ be static in a 2D plane of the motion space such that $O_i$ is the $i$-th nearest neighbor of $O_q$, as shown in Figure 44(a). Denote by $R_i$ the circle the center of which is the location of $O_q$ and the radius of which is the distance between $O_i$ and $O_q$. Let object $O_{k+1}$ move in the same 2D plane such that its route intersects $R_k$ twice but does not intersect $R_{k-1}$. Let $O_{k+2}$ have the same route as $O_{k+1}$ and move behind $O_{k+1}$ such that it enters $R_k$ after $O_{k+1}$ leaves $R_k$. Construct the same for $O_{k+3}$ and so on. Figure 44(b) shows the arrangement of the objects in the Time-SquareDistance space. It is easy to see that the number of answer-pairs is $2(n-k)+1$. □

**Corollary 5.1.** Assume that the query object and the data objects all move linearly. Then for any constant $k$, $\beta_k(n) = \Theta(n)$.
5.2.2 Number of Answer-pairs with Piecewise Linear Motion

Now we assume that objects move piecewise linearly in the motion space. That is, an object moves along a straight line with a constant speed from point $a$ to point $b$. At point $b$ the object changes its velocity-vector and moves to point $c$; there it changes the velocity-vector again and moves to point $d$, etc. Observe that in the Time-SquareDistance space in this model each object is represented by a connected sequence of parabola-segments (see Figure 45). It is easy to see that if the motion of $O_q$ has $m$ linear pieces and that of each object $O_i$ also has $m$ linear pieces, then $O_i$ has at most $2m-1$ parabola-segments in the Time-SquareDistance space.
According to Theorem 2.5 in [89], the length of the $k$-level in an arrangement of $N$ parabola-segments is $O(N^{2^\alpha(N)})$ where $\alpha$ is the functional inverse of Ackermann’s function$^{11}$. Based on this result, $\beta_k(n) = O(nm^{2^\alpha(n)})$ because there are at most $n(2m-1)$ parabola-segments totally for all objects in the Time-SquareDistance space. However, this derivation treats the parabola-segments of each individual object as independent segments. It does not utilize the fact that they are one-by-one connected. With the connectivity property taken into account, we obtain a tighter bound in the following theorem.

**THEOREM 5.2:** Assume that the query object and the data objects all move piecewise linearly, where each object can have at most $m$ linear pieces. Then $\beta_k(n) = O(nm^\alpha(n))$.

Due to the extremely fast growth of Ackermann’s function, its inverse $\alpha(n)$ grows extremely slowly, and is at most 4 for any practical value of $n$.

**PROOF IDEA:** The proof is inspired by Corollary 3.4 in [89]. The Corollary implies that the length of the $k$-level in an arrangement of $n$ piecewise linear functions, where each function has $m$ linear pieces, is $O(nm^\alpha(n))$. It can be shown that Corollary 3.4 in [89] holds for piecewise pseudo-linear functions as well. A piecewise pseudo-linear function is one that has one-by-one connected pseudo-linear pieces; two

---

$^{11}$ See [8] for the definition of Ackermann’s function.
pseudo-linear pieces intersect at most once. We then cut each parabola into two pseudo-linear pieces using its axis of symmetry. Thus the $O(nm\alpha(n))$ bound follows. □

The following proposition gives a lower bound of $\beta_k(n)$ in the piecewise linear model.

**Proposition 5.2:** Assume that the query object and the data objects all move piecewise linearly, where each object can have at most $m$ linear pieces. Then for any constant $k$, $\beta_k(n) \geq 2m(n-k)+1$.

Proposition 5.2 tells us that the bound of $O(nm\alpha(n))$ is very tight because it only adds an almost-constant factor to the lower bound.

**Proof Idea:** Similar to the linear case, we construct a feasible case in which the number of answer-pairs is linear in $nm$. For details see Appendix A. □

**Corollary 5.2:** Assume that the query object and the data objects all move piecewise linearly, where each object can have at most $m$ linear pieces. $\beta_d(n) = O(nm\alpha(n))$ and $\beta_d(n) = \Omega(nm)$.

### 5.3 Continuous $k$NN with Location Uncertainty

Our uncertainty model is a circular region, and the object may be anywhere in this region. This captures the uncertainty caused by positioning errors. It is also a result of the widely used “dead-reckoning” location update policy ([110]) which reduces the communication/energy cost for a moving object to report its location changes. In this policy, a moving object sends an update to the database when the distance between its current location and the location stored in the database exceeds a certain threshold. Thus at any point in time the actual location of the moving object can be anywhere in a circular region with the center being the location stored in the database and the radius being the threshold.

The rest of this section is organized as follows. In §5.3.1 we formalize the uncertainty model and define the semantics of continuous $k$NN query in the context of uncertainty. In §5.3.2 we discuss the number of answer-pairs when uncertainty regions move linearly. In §5.3.3 we discuss the number of answer-pairs when uncertainty regions move piecewise linearly.
5.3.1 Uncertainty Model and Continuous Possible kNN Queries

For any point \( t \) in time, each data object \( O_i \) has an uncertainty region \( U_i(t) \), which is a closed region where the location of \( O_i \) can possibly be at \( t \). \( U_i(t) \) is a circle area with the center denoted by \( C_i(t) \) and the radius denoted by \( r_i(t) \) (see e.g., [86, 87]). We assume that the radius \( r_i(t) \) is a constant, i.e., does not change with time. However, this constant may be different among objects. We write \( r_i(t) \) as \( r_i \). The location of \( O_q \) has no uncertainty because \( O_q \) knows its own location precisely. A location configuration at time \( t \) is a sequence of points \( p_1, p_2, \ldots, p_n \) such that \( p_1 \in U_1(t), p_2 \in U_2(t), \ldots, p_n \in U_n(t) \). Intuitively, a location configuration gives a possible location for each object at a particular time.

The uncertainty of location values implies that there may be more than \( k \) objects that are possibly the \( k \) nearest neighbors of \( O_q \) at a point in time. This concept is illustrated by Figure 46 for \( k=1 \) (i.e., nearest neighbor). Clearly both \( O_1 \) and \( O_2 \) are possible nearest neighbors at \( t \). \( O_3 \), on the other hand, is not a possible nearest neighbor at \( t \).

![Figure 46. Uncertainty regions, minimum possible distance, and maximum possible distance.](image_url)

Formally, an object \( O_i \) is a possible \( k \) nearest neighbor (PkNN) at time \( t \) if there exists a location configuration at \( t \) in which \( O_i \) is the 1st, or 2nd, or … \( k \)-th nearest neighbor. The PkNN set at time \( t \) is the set of all the objects that are PkNNs at \( t \). For the objects database instance shown in Figure 46, \( \{O_1, O_2\} \) is
the P1NN set at time $t$. The continuous $P_kNN$ (CP$kNN$) query requests for every point in time the $P_kNN$ set of $O_q$.

Denote by $d_{i}^{\text{max}}(t)$ the maximum possible distance between $O_i$ and $O_q$ at time $t$. $d_{i}^{\text{max}}(t) = d_i(t) + r_i$ where $d_i(t)$ is the distance between $C_i(t)$ and the location of $O_q$ at $t$. $(d_{i}^{\text{max}}(t))^2$ is a parabola in the Time-SquareDistance space as illustrated in Figure 47 by solid curves. The $(d_{i}^{\text{max}}(t))^2$ curve is referred to as the maximum square-distance curve, or simply the max-curve, of $O_i$. Similarly, the minimum possible distance between $O_i$ and $O_q$ at time $t$ is $d_{i}^{\text{min}}(t) = d_i(t) - r_i$. $(d_{i}^{\text{min}}(t))^2$ is also a parabola in the Time-SquareDistance space as illustrated in Figure 47 by dashed curves$^{12}$. The $(d_{i}^{\text{min}}(t))^2$ curve is referred to as the minimum square-distance curve, or simply the min-curve, of $O_i$. Thus, each data object $O_i$ is represented by a pair of parabolas namely $(d_{i}^{\text{max}}(t))^2$ and $(d_{i}^{\text{min}}(t))^2$.

![Figure 47. Max-curves (solid parabolas) and min-curves (dashed parabolas).](image)

Example: Consider the CP1NN query against the objects database instance shown in Figure 47. The answer-pairs are:

---

$^{12}$ Strictly speaking, the minimum possible distance is zero when $d_i(t) - r_i \leq 0$, which occurs when the location of $O_q$ at $t$ falls into the uncertainty region $U_i(t)$. In this case, the min-curve is a parabola truncated by the horizontal line $y=0$. However, for simplicity of presentation, we assume that $d_i(t) - r_i$ is greater than zero for all values of $t$. But all the results in this paper hold for the case where $d_i(t) - r_i$ may be smaller than or equal to zero.
We refer to the $k$-level in the arrangement of the max-curves as the \textit{max $k$-level}. The thick solid curve in Figure 47 shows a max 1-level.

The following proposition provides a method to determine a P$k$NN set at a given point in time. It is a standard technique used in existing work on P$k$NN query processing (see [94, 103, 104]).

\textbf{Proposition 5.3:} For any $k$, an object $O_i$ is a P$k$NN at a time $t$ if and only if the min-curve of $O_i$ is below the max $k$-level at $t$.

The intuition behind Proposition 5.3 is as follows. Suppose that the min-curve of $O_i$ is above the max $k$-level at time $t$. Consider the objects that are the max 1-level, max 2-level, $\ldots$, the max $k$-level at $t$. These $k$ objects are definitely closer to $O_q$ than $O_i$ regardless of the location configuration. Thus in this case $O_i$ is not a P$k$NN at $t$. Now suppose that the min-curve of $O_i$ is below the max $k$-level at time $t$. Then there are at least $n-k$ objects such that their max-curves are above $O_i$’s min-curve. Let it be such that for each of these $n-k$ objects its location takes its max-curve at $t$, and that the location of $O_i$ takes its min-curve at $t$. In this case $O_i$ is a $k$NN at $t$.

According to Proposition 5.3, the P$k$NN set changes if and only if a min-curve intersects the max $k$-level. Specifically, if the min-curve downwards crosses the max $k$-level at a time $t$, then $O_i$ should be added to the P$k$NN set starting from $t$. On the other hand, if the min-curve upwards crosses the max $k$-level at a time $t$, then $O_i$ should be removed from the P$k$NN set starting from $t$.

As shown in the Figure 47 example, a P$k$NN set may contain more than $k$ objects. If size of the set is much larger than $k$, then the user may request that the P$k$NNs be ranked by their probabilities of being a $k$
nearest neighbor for a particular time point. In this case, the technique presented in [86, 93] can be used for ranking.

5.3.2 Number of Answer-pairs for CPkNN Queries with Linear Motion

Denote by \( \tilde{I}_i(G) \) the number of answer-pairs for an instance \( G \) of the objects database for the CPkNN query. Define

\[
\tilde{\beta}_k(n) = \max \{ \tilde{I}_i(G) | G \text{ is an instance of the objects database with } n \text{ objects} \}
\]

In words, \( \tilde{\beta}_k(n) \) is the maximum number of answer-pairs for the CPkNN query among all the objects database instances with \( n \) objects.

We assume that the query object \( O_q \) moves linearly. In addition, for each data object \( O_i \), its expected location \( C_i(t) \) also moves linearly. It is obvious that \( \beta_k(n) \) is upper bounded by \( O(n^2) \) because there are \( O(n^2) \) intersections among all the min-curves and all the max-curves. The following theorem indicates that \( O(n^2) \) is also the lower bound of \( \beta_k(n) \).

**THEOREM 5.3.** Assume that the query object moves linearly and that each data object’s expected location moves linearly. Then for any constant \( k \) \( \tilde{\beta}_k(n) = \Omega(n^2) \).

**PROOF.** Prove by constructing a feasible case in which the number of answer-pairs is quadratic in \( n \).

The construction proceeds as follows. Let object \( O_q \) and the expected locations of \( O_1, O_2, \ldots, O_k \) be static in the motion space such that \( O_i \) has the \( i \)-th maximum possible distance to \( O_q \) (see Figure 48(a)). Denote by \( R_i^\text{max} \) the circle the center of which is the location of \( O_q \) and the radius of which is the maximum possible distance between \( O_i \) and \( O_q \). Let object \( O_{k+1} \) move such that the trace of its farthest-to-\( O_q \)-point intersects \( R_i^\text{max} \) twice but does not intersect \( R_{k+1}^\text{max} \), as shown in Figure 48(a). Let \( O_{k+2} \) have the same route as \( O_{k+1} \) and move behind \( O_{k+1} \) such that its farthest-to-\( O_q \)-point enters \( R_i^\text{max} \) after that of \( O_{k+1} \) leaves \( R_i^\text{max} \). Construct the same for \( O_{k+3}, \ldots, O_{n/2} \). Figure 48(b) shows the arrangement of the max-curves of the first \( n/2 \) objects.
Denote by $R_i^{\text{min}}$ the circle the center of which is the location of $O_q$ and the radius of which is the minimum possible distance between $O_i$ and $O_q$. Let $O_{n/2+1}$ be static such that $R_{n/2+1}^{\text{min}}$ twice intersects the trace of $O_{k+1}$’s farthest-to-$O_q$-point (see Figure 48(c)). Do the same construction for $O_{n/2+2}, \ldots, O_n$. Figure 48(d) shows how the min-curves intersect the max $k$-level. It is not difficult to see that there are totally $2 \cdot \frac{n}{2} \cdot \frac{n}{2} - k$ critical intersections. □

**Corollary 5.3.** Assume that the query object moves linearly and that each data object’s expected location moves linearly. Then for any constant $k$ $\tilde{\beta}_k(n) = \Theta(n^2)$.

Corollary 5.3 is particularly interesting when compared with the result in the certain case: the number of answer-pairs is linear in $n$ in the certain case (see Theorem 5.1) whereas it is quadratic in $n$ in the uncertain case.

(a) Configuration of the first $n/2$ objects in the motion space.
(b) Max-curves of the first $n/2$ objects.

(c) Configuration of $O_{n/2+1}$ in the motion space.

(d) Intersections between the min-curve of $O_{n/2+1}$ and the max $k$-level.

Figure 48. The auxiliary figure for the proof of Theorem 5.3
5.3.3 Number of Answer-pairs for CPkNN Queries with Piecewise Linear Motion

In this subsection we derive the upper bound and lower bound of $\tilde{\beta}_k(n)$ for the piecewise linear model. Observe that in the piecewise linear model, each max-curve is a connected sequence of parabola curves, and so is each min-curve. The following theorem gives an upper bound of $\tilde{\beta}_k(n)$.

**Theorem 5.4.** Assume that the query object and the expected locations of each data object move piecewise linearly, where each object has at most $m$ pieces. Then for any constant $k$, 
$$\tilde{\beta}_k(n) = O(n^2 \cdot m \cdot \alpha(n)).$$

The proof of Theorem 5.4 is based on the following lemma which bounds the number of intersections between two connected sequences of parabola pieces. A connected sequence of parabola pieces is a sequence of parabola pieces such that the ending point of one piece is the starting point of its successive piece. An obvious bound for the number of intersections between two sequences parabola pieces, where one sequence has $m_A$ pieces and another sequence has $m_B$ pieces, is $O(m_A \cdot m_B)$. However, the following lemma gives a much tighter bound which is $O(m_A + m_B)$.

**Lemma 5.1:** Let $A$ be a connected sequence of $m_A$ x-monotone parabola-pieces and $B$ be a connected sequences of $m_B$ x-monotone parabola-pieces, in the Time-SquareDistance space. There are at most $2(m_A + m_B - 1)$ intersections between $A$ and $B$.

The proof of Lemma 5.1 is provided in Appendix B. Now we prove Theorem 5.4.

**Proof of Theorem 5.4:** According to Theorem 5.2, the length of the max $k$-level is $O(n \cdot m \cdot \alpha(n))$.

Since each min-curve has at most $2m-1$ pieces, according to Lemma 5.1, the number of intersections between each min-curve and the max $k$-level is upper bounded by

$$2 \cdot ((2 \cdot m - 1) + O(n \cdot m \cdot \alpha(n)) - 1)$$

$$= O(n \cdot m \cdot \alpha(n))$$

Thus the total number of intersections between all the min-curves and the max $k$-level is

$$n \cdot O(n \cdot m \cdot \alpha(n)) = O(n^2 \cdot m \cdot \alpha(n)).$$

$\square$
The following theorem gives a lower bound of $\tilde{\mathcal{B}}_k(n)$.

**THEOREM 5.5.** Assume that the query object and the expected locations of each data object move piecewise linearly, where each object has at most $m$ pieces. Then for any constant $k$ $\tilde{\mathcal{B}}_k(n) = \Omega(n^2 \cdot m)$.

We prove the theorem by constructing a feasible case in which the number of answer-pairs is in order of $n^2 \cdot m$. For details see Appendix C.

### 5.4 Query Processing without uncertainty

In this section we discuss the processing of $C_k$NN queries without uncertainty. In §5.4.1 we discuss off-line processing and in §5.4.2 we discuss on-line processing.

#### 5.4.1 Off-line Processing

When $k=1$, the processing of a $C_k$NN query reduces to the construction of the lower envelope in the arrangement of the square-distance functions of the $n$ data objects (i.e., the $d_j^2(t)$’s). In the linear model, the lower envelope can be constructed using a simple and efficient divide-and-conquer algorithm in $O(n \log n)$ time (see Theorem 2.6 in [89]). This algorithm divides the objects database into two subsets $S_1$ and $S_2$, each of size at most $\lceil n/2 \rceil$, computes the lower envelopes of $S_1$ and $S_2$ recursively, and merges the two envelopes to obtain the lower envelope of the objects database.

The same paradigm can be used to compute the lower envelope in the piecewise linear model. In this case, merging the lower envelopes of $S_1$ and $S_2$ still takes time proportional to the sum of $|S_1|$ and $|S_2|$ as described in [89]. Due to Theorem 5.2, $|S_1|+|S_2|=O(nma(n)).$ Thus the complexity in the piecewise linear model is $T(n) = 2T\left(\frac{n}{2}\right) + O(nma(n))$, which is $O(nma(n)\log n)$.

When $k>1$, the answer-pairs can be computed in the same fashion. Details are omitted due to space limitations.
5.4.2 **On-line Processing**

In §5.4.2.1 we describe an existing on-line processing data structure called *kinetic tournament* and discuss its shortcoming. From 4.2.2 to 4.2.5 we present our on-line processing data structure which overcomes this shortcoming.

5.4.2.1 **Kinetic Tournament**

Consider the case in which \( k = 1 \) and all the objects move linearly. The query processing in this case translates to the problem of dynamically maintaining the lower envelope of a set of parabolas in the Time-SquareDistance space. In computation geometry this problem is known as *kinetic minimum maintenance*. The authors of [91] introduce a solution to this problem, which is called a *kinetic tournament*. The idea is to use a simple divide-and-conquer strategy. The algorithm partitions the data objects into two approximately equal-sized groups (arbitrarily), and recursively maintains the minimum of each group. A final comparison at the top level yields the global minimum. If viewed from the bottom up, this is exactly a tournament for computing the global winner. Each comparison of two data objects is associated with an event that describes when this comparison will be violated in the future. When a violation happens, the new winner is produced and is percolated up the tournament tree, until it is either defeated or declared the global winner.

Figure 49 gives an example of kinetic tournament. Figure 49(a) shows three data objects \( O_1, O_2, \) and \( O_3 \) in the Time-SquareDistance space. Figure 49(b) shows the tournament tree constructed at time 0. Two events are scheduled at this moment: (i) \( O_1 \) crosses \( O_2 \) at time \( t_1 \) which represents that \( O_1 \) beats \( O_2 \) up to \( t_1 \); (ii) \( O_1 \) crosses \( O_3 \) at time \( t_2 \) which represents that \( O_1 \) beats \( O_3 \) up to \( t_2 \). Since \( t_1 < t_2 \), the \( O_1 \)-crossing-\( O_2 \) event is triggered earlier, at \( t_1 \). In response to this event, the winner between \( O_1 \) and \( O_2 \) is changed, and the new winner is percolated up (see Figure 46(c)). The \( O_1 \)-crossing-\( O_3 \) event is eliminated, and two new events are scheduled to be triggered, at time \( t_4 \) and \( t_3 \) respectively.
Now examine the complexity of kinetic tournament on processing each event. The processing of an event involves the percolation of the new winner. Because a tournament tree is balanced, the percolation visits at most $O(\log n)$ nodes. A visit to each of these nodes may result in elimination of an existing event and creating of a new event. If we use a priority queue to store the relevant events (at most $O(n)$), then the elimination or creation of an event takes $O(\log n)$ time. Thus, the percolation takes $O(\log^2 n)$ time.

As analyzed above, in kinetic tournament, $O(\log n)$ time is spent on accessing the event queue when processing an event. Indeed, kinetic tournament schedules an event for each internal node in the tournament tree, even though some of these events will never be triggered. In the Figure 49 example, the $O_1$-crossing-$O_3$ event is never triggered because it becomes useless after $O_2$ claims the global winner. In fact we only need to schedule one event which is the earliest time at which the current global winner may be replaced. In this way we eliminate the cost of accessing the event queue. This is the main idea of our on-line processing data structure which is presented in the following subsection.
5.4.2.2 The Object Heap Data Structure

Our kinetic data structure is called object heap and is organized as follows. As before, we assume that we have a query object $O_q$ and a set $S$ of data objects such that $O_q \notin S$. We assume that there is a unique id with each object in $S$. Let $t \geq 0$ be a time instance. An object heap $H$ over the set $S$ at time $t$ is a full binary tree such that each node $x$ in it stores two items $x.object$ and $x.time$ satisfying the following conditions.

First, for any internal node $x$, let $ObjectSet(x)$ denote the set of objects stored at the leaves of the sub-tree rooted at $x$. For a leaf node $x$, $x.time = \infty$. For an internal node $x$, $x.time \geq t$ and $x.object$ is the closest object to $O_q$ during the time interval $[t, x.time]$ among all the objects in $ObjectSet(x)$. For any internal node $x$, we let $x.left$, $x.right$ represent respectively the left and right child of $x$. Also, $x.parent$ represents the parent of $x$. Figure 50 shows the object heap at time 0 for the arrangement of Figure 49(a). From the heap we see that $O_1$ is the closest object until $t_1$.

![Figure 50. The object heap at time 0 for the arrangement of Figure 49(a).](image)

An object heap is constructed using the procedure $BuildObjectHeap(S,O_q,Ctime)$ given below. The procedure takes the set of objects $S$, the query object $O_q$ and time $Ctime$ and builds an object heap at time $Ctime$. $CloserObject(O_1,O_2,t)$, $NextTime(O_1,O_2,t)$ are functions that take two objects $O_1$, $O_2$ and time $t$ as parameters and returns values as defined below. $CloserObject(O_1,O_2,t)$ returns the object closest to $O_q$ among the two objects $O_1$, $O_2$ at time $t$. $NextTime(O_1,O_2,t)$ returns the earliest time $t' > t$ such that
$\text{CloserObject}(O_1, O_2, t')$ is different from $\text{CloserObject}(O_1, O_2, t)$, i.e. when one of the two objects becomes closer than the other; if no such $t'$ exists then it returns $\infty$. The BuildObjectHeap procedure first initializes the heap so that each object in $S$ is stored at a leaf node. Then it processes nodes in decreasing order of their levels. For each internal node, it sets its object to be the closest object to $O_q$ at time $Ctime$ among the objects stored at its two children; it sets its time to be the minimum of the time values in its children and the next time when one of them is going to overtake the other to become closer to $O_q$. In this procedure, $\text{min}$ is a function that returns the minimum of three values.

\begin{verbatim}
BuildObjectHeap(S, O_q, Ctime)
Initialize();
For each node $x$ in decreasing values of the level of $x$
    If $x$ is an internal node
        $O_1 := x.left.object;$
        $O_2 := x.right.object;$
        $x.object := \text{CloserObject}(O_1, O_2, Ctime);$;
        $x.time := \text{min}(x.left.time, x.right.time,$
                       $\text{NextTime}(O_1, O_2, Ctime))$
\end{verbatim}

It is fairly obvious to see that the worst case time complexity of $\text{BuildObjectHeap}()$ is $O(n)$ where $n$ is the number of objects in $S$. Assuming that the start time is zero, the initial object heap is built by invoking the above procedure with the parameter value of $Ctime$ set to zero. It is to be noted that if $r$ is the root node of the object heap then $r.object$ is the closest object until the time $r.time$. Thus, at time $r.time$, the object heap needs to be readjusted. We call such a readjustment as an implicit update.
5.4.2.3  Algorithm for Implicit Updates

We classify the internal nodes of an object heap as cross nodes and minimal nodes as follows. An internal node $x$ is called a cross node, if $x.time$ is less than both $x.left.time$ and $x.right.time$. All internal nodes other than cross nodes are called minimal nodes. If $x$ is a cross node and $O_1, O_2$ are the objects at its children and $x.object = O_1$, then up to the time $x.time$ object $O_1$ is the closest to $O_q$ among objects in $ObjectSet(x)$ and at time $x.time$, $O_2$ will be the closest object among these objects. If $x$ is a minimal node, then there is a child $y$ of $x$ such that $x.time = y.time$.

Roughly speaking, the algorithm for implicit update works as follows. If the root node $r$ is a cross node then it sets $r.object$ to be the object, among its children, that is different from the current value of $r.object$ and sets $r.time$ to be the minimum of the times at its children. If the root node $r$ is a minimal node then it traverses along a path of internal nodes $x_1, ..., x_g$ such that $x_1 = r$, $x_g$ is a cross node and all nodes $x_1, ..., x_{g-1}$ are minimal nodes and the time value on all these nodes is $r.time$. After reaching $x_g$, it updates the object and time value at this node, and retraces the path back to the root node updating the object and time values on each of these nodes appropriately. The object heap of Figure 51 results when we perform implicit update on the object heap of Figure 50 at time $t_1$. The modified object heap shows that object $O_2$ is the closest object from time $t_1$ up to time $t_2$.

![Figure 51. The object heap resulting from an implicit update on the object heap of Figure 50.](image)
The above algorithm is accomplished by the recursive procedure $ImplicitUpdate(x)$ given below. This
procedure acts as follows. If $x$ is a cross node and $O_1, O_2$ are objects stored in its two children and
$x.object = O_1$ then it sets $x.object$ to $O_2$ and sets $x.time$ to be the minimal of the times of its children and
returns. Otherwise, it recursively invokes the procedure on a child $y$ such that $y.time = x.time$. This
recursive invocation may change the object and time at node $y$. Thus, when invocation on $y$ returns, it
reevaluates which of the objects at its children is the closest object and sets $x.object$ to that object and
resets $x.time$ and returns. Note that the actual implicit update is carried out by invoking this procedure
with the root $r$ at time $r.time$.

$ImplicitUpdate(x)$

\[ y := x.left; z := x.right; \]

\[ \text{If } x.time < y.time \text{ and } x.time < z.time \text{ (if $x$ is a cross node)} \]

\[ \text{If } x.object = y.object \]

\[ x.object := z.object; \]

\[ \text{Else } x.object := y.object; \]

\[ x.time := \text{NextTime}(y, z, x.time); \]

\[ \text{return;} \]

\[ \text{If } x.time = y.time \]

\[ ImplicitUpdate(y); \]

\[ \text{If } x.time = z.time \]

\[ ImplicitUpdate(z); \]

\[ x.object := \text{CloserObject}(y, z, x.time); \]

\[ x.time := \min(y.time, z.time, \text{NextTime}(y, z, x.time)); \]
Note that for an object heap at time \( t \), with root \( r \), \( t < r.time \), \( r.time \) is less than or equal to \( x.time \) for every node \( x \) in it, and the structure satisfies the heap property until \( r.time \). At time \( r.time \), we invoke the procedure \( ImplicitUpdate(r) \). It is fairly straightforward to show that after invocation of \( ImplicitUpdate(r) \) at time \( r.time \), the structure continues to be an object heap.

We make the assumption, called *distinct-sibling-times* assumption, that for every pair of internal nodes \( y \), \( z \), that are siblings, \( y.time \neq z.time \). At the end of this subsection we will discuss what happens when this assumption does not hold. Under the distinct-sibling-times assumption, when \( ImplicitUpdate(r) \) is invoked, the algorithm will travel along a single path in the object heap. Hence, the complexity of executing \( ImplicitUpdate(r) \) is \( O(h) \) where \( h \) is the height of the object heap. Since \( h \leq \log n + 1 \), we see that the complexity of execution of the ImplicitUpdate is \( O(\log n) \). Furthermore, at any point in time there is only one event to monitor which is the expiration of the root node. Now the following theorem shows that there can be at most \( O(n\log n) \) implicit updates, i.e., after at most \( n\log n \) updates, \( r.time = \infty \) and hence the object heap does not need any further implicit updates. Thus the object heap is efficient in the sense that the number of implicit updates is only \( \log n \) larger than the maximum number of times that the answer set may change.

**Lemma 5.2:** Let \( T \) be an object heap at time 0 with root \( r \), then after at most \( 2n\log n \) updates, \( r.time = \infty \).

**Proof:** Observe that whenever an implicit update is performed, the object heap is traversed starting from the root \( r \) until a cross node \( x \) is reached. At the cross node \( x \), the value of \( x.object \) and \( x.time \) are updated. It should be easy to see that the time when this update is carried out, i.e., at this time \( x.time = r.time \), and at this time the closest object to \( O_q \) among \( ObjectSet(x) \) changes and hence it is a split point for this set of objects. Since there are at most \( 2n_1-1 \) split points for \( ObjectSet(x) \) where \( n_1 \) is the number of objects in \( ObjectSet(x) \), we see that the number of implicit updates on \( T \) that stop at \( x \) is bounded by \( 2n_1-1 \). Let \( l \) be the level number of \( x \). Now the total number of implicit updates that stop at a
node at level $l$ is bounded by twice the number of objects stored in the sub-trees whose root is at level $l$. Since the number of all such objects is the total number of objects, we see that the total number of implicit updates that stop at a level $l$ node is bounded by $2n$. Since there are $\log n$ levels, we see that the total number of implicit updates is bounded by $2n \log n$. Clearly, after at most such implicit updates $r.time=\infty$. □

Since there are $O(n \log n)$ implicit updates and each such update takes time $O(\log n)$, we see that the cumulative complexity of performing these updates is $O(n \log^2 n)$.

If the distinct-sibling-times assumption does not hold then an invocation of $\text{ImplicitUpdate}(r)$ may travel on multiple paths, and hence the complexity may be higher than $O(h)$; however, its complexity can be shown to be only $O(n)$. Furthermore, the cumulative complexity over a number of invocations until $r.time=\infty$, can easily be shown to be still $O(n \log^2 n)$.

5.4.2.4 Explicit Updates and Piecewise Linear Model

Now we show how explicit updates can be implemented efficiently using object heap. We consider three types of updates, namely insertion of a new object, deletion of an existing object, and velocity-vector change of an existing object.

**Addition.** Assume that we have an object heap $H$ with root node $r$ storing $n$ objects. In this heap which has $2n^2 - 1$ nodes, each leaf node will have a sibling. Assume that the new object to be inserted is $O'$ at time $Ctime$ which is the current time. We can assume that $Ctime < r.time$. We allocate two new nodes, say $y_1$ and $y_2$. Let $x$ be the first leaf node in $H$, i.e., the leftmost leaf node at the lowest numbered level (note that if the height of $H$ is $h$, then there can be leaf nodes both at level $h^2 - 1$ and $h$). Also let $O''$ be the object in node $x$. Now we add the two nodes $y_1, y_2$ as children of $x$ and place objects $O'$ and $O''$ in these two nodes. This makes $x$ as an internal node. Now we perform, the following operation, called $\text{float}(z)$ with argument $x$. This operation $\text{float}(z)$, is a recursive operation, which acts as follows. It sets $z.object$ to the object among its children that is closest to the query object $O_q$ at time $Ctime$, i.e., to the object given by the
function \textit{CloserObject}(O_1, O_2, Ctime) and sets \textit{z.time} to be the value given by the function \textit{NextTime}(O_1, O_2, Ctime) where \(O_1, O_2\) are the objects \(z\_left\_object\) and \(z\_right\_object\), respectively. After this, if \(z\) is the root then it stops; else it recursively invokes \textit{float} on the parent of \(z\). It is not difficult to see that the resulting structure \(H\) is an object heap at time \(Ctime\) containing the \(n+1\) objects. It's complexity is clearly \(O(\log n)\).

Consider the object heap of Figure 50. Assume that \(t_i > 1\). When we insert a new object \(O_4\) at time 1, the following heap results (see Figure 52). Note that \(O_1\) is still the closest object till \(t_1\). Also \(O_3\) is the closer of \(O_3\) and \(O_4\) until \(t_5\). Here \(t_5\) is assumed to be greater than \(t_1\).

\[\begin{array}{c}
\text{Figure 52. The object heap resulting from an explicit update on the object heap of Figure 50.}
\end{array}\]

\textbf{Deletion.} Now we describe the deletion of an object from \(H\). Assume that the object to be deleted is the object in the last leaf node, i.e. the right most node in the lowest level (i.e., at level \(h\)). Let \(y_2\) be this node and let its sibling be \(y_1\) and let \(x\) be their parent. It is not difficult to see that \(x\) is the last internal node of \(H\). We delete both the nodes \(y_1\) and \(y_2\) and place the object in \(y_1\) in the node \(x\). By this, \(H\) loses two nodes and \(x\) becomes a leaf node. Now, we invoke \textit{float} operation on the parent \(z\) of \(x\). After this, the resulting structure will satisfy the object heap property. Clearly, the complexity of this operation is \(O(\log n)\).
Now consider the deletion of an object \( O' \) in an arbitrary leaf node \( y \). To do this, we first delete the object \( O'' \) in the last leaf node of \( H \) using the above procedure. In the resulting object heap, let \( y' \) be the leaf node containing object \( O' \). Notice that it is possible that \( y' \) is different from \( y \). This occurs if \( y \) was the last but one leaf node in \( H \). In any case, we replace object \( O' \) in \( y' \) by \( O'' \) and perform the \textit{float} operation starting from the parent of \( y' \). It should be easy to see that the resulting structure is an object heap containing all objects of \( H \) excepting \( O' \). Clearly, the complexity of this whole procedure is \( O(\log n) \).

\textbf{Velocity-vector Change.} When the speed and/or the direction of an existing object \( O' \) changes, the square-distance function \( d_{O'}^2(t) \) changes. In this case, find the leaf node \( y \) that contains \( O' \) and update \( d_{O'}^2(t) \). After this, we perform the \textit{float} operation starting from the parent of \( y \). The complexity of this update is also \( O(\log n) \).

Now we compare object heap and two existing algorithms in terms of their scalability to database size \( n \). The two existing algorithms are kinetic tournament and off-line processing. Recall from §5.4.2.1 that the complexity of an update in kinetic tournament is \( O(\log^2 n) \). Recall from §5.4.1 that the complexity of off-line processing \( O(n \log n) \). This is also the complexity of off-line processing to process an update because for each update off-line processing has to re-compute all the answer-pairs. On the other hand, the above analysis shows that the complexity of an update in object heap is \( O(\log n) \). The implication of this is that for a very large database, object heap is better by a factor of \( \log n \), compared to kinetic tournament, in the worst case. Furthermore, object heap is better by a factor of \( n \), compared to off-line processing, in the worst case. Thus object heap is more scalable to large databases.

Finally let us consider the piecewise linear model in which each object can change its velocity-vector at most \( m \) times and we analyze the cumulative complexity of our on-line algorithm for maintaining the 1NN set and handling velocity-vector changes. The cumulative complexity is bounded as follows. Since the complexity for handling each velocity-vector update is \( O(\log n) \), the total complexity of these updates is \( O(nm \log n) \). Note that implicit updates are carried out between velocity-vector changes as
before, as and when needed. Now, we bound the cumulative complexity of implicit updates. Observe that each object has a piecewise linear motion with at most m pieces throughout the time. Using Theorem 5.2, it is not difficult to see that the total number of implicit updates that stop at a particular internal node \( x \) is \( O(n_1 m \alpha(n_1)) \) where \( n_1 \) is the number of objects stored in the sub-tree rooted at \( x \). Using the same argument as given in Lemma 5.2 for the linear case, we see that the total number of implicit updates that stop at an internal node at some level say \( l \) is \( O(n m \alpha(n)) \). Since there are at most \( \log n \) levels, we see that the total number of implicit updates is \( O(n m \alpha(n) \log n) \). Since the cost of each implicit update is \( O(\log n) \), we see that the cumulative time complexity of implicit updates is \( O(n m \alpha(n) \log^2 n) \). This is also the cumulative complexity of all the updates both implicit as well as explicit.

5.4.2.5 **Extension to \( k > 1 \)**

For \( k > 1 \), each internal node \( x \) in the object heap stores two items \( x.set \) and \( x.time \) such that \( x.set \) is the set of the \( k \) closest objects to \( O_q \) up to time \( x.time \) among all the objects in \( ObjectSet(x) \). To achieve this goal, we modify the procedure \( BuildObjectHeap(S, O_q, Ctime) \) as follows. For each internal node \( x \), \( x.set \) is set to be the \( k \)NN set at time \( Ctime \) among the objects stored at its two children; \( x.time \) is set to be the minimum of the time values in \( x \)'s children and the next time when the \( k \)NN set among the objects stored at \( x \)'s two children is going to change.

Now consider how to process an implicit update that is triggered at time \( t \). Similar to \( k=1 \) case, the implicit update algorithm traverses along a path of internal nodes \( x_1, \ldots, x_g \) such that \( x_1=r \), \( x_g \) is a cross node and all nodes \( x_1, \ldots, x_{g-1} \) are minimal nodes and the time value on all these nodes is \( r.time \). After reaching \( x_g \), it sets \( x_g.set \) to be the \( k \)NN set at time \( t \) among the objects stored at \( x_g \)'s two children and sets \( x_g.time \) to be the minimum of the time values in \( x_g \)'s children and the next time when \( x_g.set \) is going to change.

Explicit updates can be extended in the same fashion. It is not difficult to see that, when \( k \) is a constant, the complexity of our on-line algorithm for \( k > 1 \) is the same as that for \( k=1 \).
5.5 **Query Processing with Uncertainty**

In this section we discuss the processing of CPkNN queries. In §5.5.1 we discuss off-line processing and in §5.5.2 we discuss on-line processing.

5.5.1 **Off-line Processing**

The off-line algorithm is based on Proposition 5.3 introduced in §5.3.1. According to this proposition, the PkNN set changes if and only if a min-curve intersects the max k-level. Thus, the algorithm first constructs the max k-level, then it finds the intersections between all the min-curves and the max k-level, and finally it determines the PkNN set created by each of these intersections. Specifically, the algorithm has the following steps:

Step 1: Compute the max k-level using the algorithm discussed in §5.4.1. The output of this step is the list of the edges in the max k-level sorted in ascending order of the time attribute of their start points.

Step 2: For each data object $O_i$, compute the intersections between $O_i$’s min-curve and the max k-level. Call these intersections *critical intersections*. Denote the time coordinate of a critical intersection $p$ by $p_{time}$. Since the edges in the max k-level are already sorted, the critical intersections of $O_i$ are automatically sorted. For each intersection $p$ of these intersections, compute $D$ the first order derivative of $O_i$’s min-curve at $p$ and $D'$ the first order derivative of the max k-level at $p$. If $D$ is smaller than $D'$, we say that $p$ is an *entry point* and let $p_{entry}=i$. Otherwise, we say that $p$ is an exit point and let $p_{exit}=i$. Intuitively, if $p$ is an entry point, then $O_i$ becomes a PkNN starting from $p_{time}$; if $p$ is an exit point, then $O_i$ stops being a PkNN starting from $p_{time}$.

Step 3: Sort the critical intersections of all the data objects in ascending order of their time attribute. Since the sequence of critical intersections is already sorted for each $O_i$, we use the multi-way merge algorithm introduced in [115] to merge these sequences to a global sequence. The algorithm uses a priority queue to store the smallest element and has complexity $O(M\log N)$ where $N$ is the number of input sequences and $M$ is the total number of elements in all input sequences.
Step 4: Find the set of objects whose min-curves are below the max $k$-level at time 0. Denote this set by $S_1$. Let $T_0=0$.

Step 5: For each critical intersection $p_i$, let $T_i=p_i.time$. Furthermore, if $p_i$ is an entry point, then let $S_{i+1}=S_i\cup\{p_i.entry\}$; otherwise, let $S_{i+1}=S_i-\{p_i.entry\}$.

Step 6: Let $T_h=\infty$ where $h$ is the total number of critical intersections.

It is easy to see that the two output sequences given by $S_1<S_2<\ldots<S_h$ and $T_1,T_2,\ldots,T_h$ satisfy the property that $T_h=\infty$, and for each $l, 1\leq l\leq h$, $S_l$ is the $P_k$NN set during the interval $[T_{l-1},T_l)$ and no two successive object sets in the $S$-sequence are identical.

Now we analyze the complexity of the above algorithm. In the linear model, the complexity of Step 1 is $O(n\log n)$ as discussed in §5.4.1. The complexity of Step 2 is $\Theta(n^2)$ according to Corollary 5.3. The complexity of Step 3 is $O(n^3\log n)$ for multi-way merging. The complexity of Step 4 is $n$. The complexity of Step 5 is $O(n^2)$. Thus the complexity of the off-line algorithm in the linear model is $O(n^2\log n)$. Observe that the number of answer-pairs is $\Omega(n^2)$ (see Theorem 5.3), which gives a lower bound for the cost of any off-line algorithm. Our algorithm is only $\log n$ higher than this lower bound.

In the piecewise linear model, the complexity of Step 1 is $O(nm\alpha(n)\log n)$ as discussed in §5.4.1. The complexity of Step 2 is analyzed as follows. According to Theorem 5.2, there are $O(nm\alpha(n))$ parabola pieces in the max $k$-level. Each min-curve has $O(m)$ parabola pieces. According to Lemma 5.1, there are $O(nm\alpha(n))$ intersections between each min-curve and the max $k$-level. Thus there are totally $O(n^2m\alpha(n))$ intersections between $n$ min-curves and the max $k$-level. The complexity of Step 3 is $O(n^2m\alpha(n)\log n)$ for multi-way merging. The complexity of Step 4 is $n$. The complexity of Step 5 is $O(n^2m\alpha(n))$. Thus the complexity of the off-line algorithm in the piecewise linear model is $O(n^2m\alpha(n)\log n)$.

An alternative approach to off-line processing with uncertainty is a divide-and-conquer algorithm that is similar to the one discussed §5.4.1 for the certain case. It can be shown that the complexity of this approach is also $O(n^2\log n)$ for the linear model and $O(n^2m\cdot\alpha(n)\cdot\log n)$ for the piecewise linear model.
5.5.2 **On-line Processing**

For on-line processing, we need to track the change of the $P_k$NN set. Recall that the $P_k$NN set changes whenever a min-curve intersects the max $k$-level. Thus we monitor two types of events. One is the change of the max $k$-level, called *max $k$-level events*, and another is critical intersections (i.e., the intersections between min-curves and the max $k$-level), called *intersection events*. At a high level, our on-line processing algorithm works as follows. We adapt the object heap data structure introduced in §5.4.2 to monitor max $k$-level events and use a priority queue to monitor intersection events. When an intersection event is triggered, an answer-pair is produced as an output. When a max $k$-level event is triggered, intersections between min-curves and the new max $k$-level are computed and the corresponding intersection events are scheduled. In §5.5.2.1 we present our on-line processing algorithm for $k=1$ for the linear model. In §5.5.2.2 we extend this algorithm to the piecewise linear model. In §5.5.2.3 we discuss the extension to $k>1$.

5.5.2.1 **The Linear Model**

In order to monitor max 1-level events, we adapt the object heap data structure as follows. We redefine the function $\text{CloserObject}(O_1, O_2, t)$ such that it returns the object that has the smallest maximum possible distance to $O_q$, among $O_1$ and $O_2$, at time $t$. We call the object heap constructed with this definition the *max object heap*, and denote it by $MOH$. It is easy to see that if $r$ is the root node of $MOH$ then $r.object$ is the object that has the smallest maximum possible distance until the time $r.time$. Implicit updates to $MOH$ are performed in the same way as to the object heap. Each implicit update causes either of the following two changes to the root node $r$: (i) both $r.object$ and $r.time$ change; and (ii) $r.object$ does not change but $r.time$ increases. Each implicit update, after its completion, triggers a max 1-level event. The max 1-level event is processed by the procedure $\text{Max1LevelUpdate}(r, t)$ described below. Compute $P$ the set of critical intersections that will occur between time $t$ and $r.time$. For each critical intersection $p \in P$, create an intersection event $E=\langle p, p.time \rangle$ which specifies that $E$ is to be triggered at $p.time$ and it will cause a change to the $P1NN$ set due to the critical intersection $p$. Build a priority queue
of the created events such that the event with the earliest trigger time is the head. Call this priority queue the event queue.

The next event that is going to be triggered is either the max 1-level event, or the head of the event queue, whichever occurs earlier. When the $i$-th intersection event $E$ is triggered, it is removed from the event queue and processed as follows. First, the P1NN set is updated. Specifically, let $S_i$ be the P1NN set before the $i$-th event is triggered and let $E = \langle p, \text{time} \rangle$. If $p$ is an entry point (i.e., an object enters the answer set at $p$.time), then let $S_{i+1} = S_i \cup \{p.\text{entry}\}$. If $p$ is an exit point (i.e., an object exits the answer set at $p$.time), then let $S_{i+1} = S_i - \{p.\text{exit}\}$. Second, let $E'$ be the first event in the remaining event queue. If the remaining event queue is empty, let $E'$ be the root of MOH. Output the pair $(S_{i+1}, E'.\text{time})$ which indicates that the P1NN set is $S_{i+1}$ from now on until $E'.\text{time}$. Notice that by the time the next implicit update occurs, the event queue will be empty.

The above process is repeated until $r$.time is $\infty$.

Now we study the complexity of our on-line processing algorithm. First, consider the complexity for processing each individual event. The cost for processing an intersection event is $O(\log n)$ for removing the event from the event queue which has $O(n)$ events. The cost for processing a max 1-level event involves executing the Max1LevelUpdate procedure. This procedure computes critical intersections and builds a priority queue of intersection events. There are $O(n)$ critical intersections induced by each max 1-level event, and thus the cost of the Max1LevelUpdate procedure is $O(n)$ for computing $O(n)$ critical intersections and building a priority queue of $O(n)$ intersection events. Observe that there are only $O(n \log n)$ max 1-level events in comparison with $O(n^2)$ intersection events which only require $\log n$ time each.

Now consider the total number of events that need to be processed. The total number of intersection events is $\Theta(n^2)$ according to Corollary 5.3. The total number of max 1-level events is $O(n \log n)$ according to Lemma 5.2. Thus the total number of events that need to be processed is $O(n^2)$ which is of the same order as the number of times the P1NN set changes.
Finally, let us analyze the cumulative complexity of our on-line processing algorithm. The cumulative cost for processing intersection events is $O(n^2 \log n)$. The cumulative cost for processing implicit updates is $O(n \log^2 n)$ as discussed in §5.4.2.3. The cumulative cost for processing max 1-level events is $O(n^2)$. Thus the cumulative complexity of our on-line processing algorithm is $O(n^2 \log n)$. Observe that the number of answer-pairs is $\Omega(n^2)$ (see Theorem 5.3), which gives a lower bound for the cost of any on-line algorithm for maintaining the P1NN set. Our algorithm is higher by only a factor of $\log n$ than this lower bound.

5.5.2.2 Explicit Updates and Piecewise Linear Model

**Addition.** Consider the addition of an object $O'$ at a time point $t'$. The first step is to add $O'$ to MOH in $O(\log n)$ time as described in §5.4.2.4. Depending on how the root node $r$ changes as a result of the addition, there are three cases for the second step.

Case 1: The max-curve of $O'$ is always farther from $O_q$ than that of $r.object$ between $t'$ and $r.time$. In this case, neither $r.object$ nor $r.time$ changes. To process this case, compute the intersections between the min-curve of $O'$ and the max-curve of $r.object$ that will occur between $t'$ and $r.time$. Create an intersection event for each of these intersections (at most two) and insert it to the event queue.

Case 2: The max-curve of $O'$ switches the distance order with that of $r.object$ some time after $t'$ and before $r.time$. In this case, $r.object$ does not change but $r.time$ decreases. This case is illustrated by Figure 53, in which $r.time$ decreases from a farther time to $t''$ due to the addition of $O'$. In this case, the intersection events in the event queue that will occur after the new $r.time$, such as intersection $p$ in Figure 53, are no longer valid. However, we leave them in the event queue. They will be eliminated by the max 1-level event triggered at $r.time$.

Case 3: The max-curve of $O'$ is currently closer to $O_q$ than that of $r.object$ and therefore $r.object$ changes. In this case, eliminate the existing event queue; invoke the procedure Max1LevelUpdate($r$, $t'$) to compute and schedule new intersection events.

The complexity of the addition is dominated by Case 3 which is $O(n)$. 
Deletion. Consider the deletion of an object $O'$ at a time point $t'$. The first step is to delete $O'$ from $MOH$ in $O(\log n)$ time as described in §5.4.2.4. Depending on how the root node $r$ changes as a result of the deletion, there are three cases for the second step.

Case 1: The max-curve of $O'$ is always farther from $O_q$ than that of $r.object$ between $t'$ and $r.time$. In this case, neither $r.object$ nor $r.time$ changes. To process this case, remove the critical intersections incurred by $O'$ if any from the event queue.

Case 2: The max-curve of $O'$ switches the distance order with that of $r.object$ some time after $t'$ and before $r.time$. In this case, $r.object$ does not change but $r.time$ increases. This case is illustrated by Figure 53, in which $r.time$ increases from $t''$ to a farther time due to the deletion of $O'$. In this case, some new intersection events may need to be scheduled, such as intersection $p$ in Figure 53. Observe that there may be $O(n)$ such intersection events. Inserting them to the existing event queue takes $O(n \log n)$ time. On the other hand, eliminating the existing event queue and reconstructing a new one takes only $O(n)$ time. Thus, in Case 2 we eliminate the existing event queue; invoke the procedure Max1LevelUpdate($r$, $t'$) to compute and schedule new intersection events.

Case 3: $O'$ is currently $r.object$ and therefore $r.object$ changes. In this case, eliminate the existing event queue; invoke the procedure Max1LevelUpdate($r$, $t'$) to compute and schedule new intersection events.

The complexity of the deletion is dominated by Case 2 and Case 3 which is $O(n)$ each. Thus the complexity of the deletion is $O(n)$. 
Velocity-vector Change. Consider that an object $O'$ changes its velocity-vector at time $t'$. The first step is to perform velocity-vector update to the object heap as described in §5.4.2.4. Depending on how the root node $r$ changes as a result of the update to the object heap, there are three cases for the second step:

Case 1: Both old($O'$) and new($O'$) are always farther from $O_q$ than the max-curve of $r.object$ between $t'$ and $r.time$, where old($O'$) and new($O'$) represent the max-curves of $O'$ before and after the velocity-vector change, respectively. In this case, neither $r.object$ nor $r.time$ changes. To process this case, compute the intersections between the new min-curve of $O'$ and the max-curve of $r.object$ that will occur between $t'$ and $r.time$. Create an intersection event for each of these intersections (at most two) and insert it to the event queue.

Case 2: $r.object$ does not change but $r.time$ decreases. This case may happen when $O'$ is $r.object$, i.e., it is $r.object$ that changes its velocity-vector. Case 2 may also happen when $O'$ is not $r.object$, as illustrated in Figure 54. In the figure, $r.time$ decreases from a farther time to $t''$ due to the velocity-vector change of $O'$ which is not $r.object$. No further processing is needed in this case.

Case 3: $r.object$ does not change but $r.time$ increases. Again this case may happen when either $r.object$ or some other object changes velocity-vector. In this case, eliminate the existing event queue and invoke the procedure Max1LevelUpdate($r, t'$).
The complexity of processing each velocity-vector update is dominated by Case 3 which is $O(n)$.

![Diagram](image)

**Figure 54.** *r.time* may change due to the velocity-vector change of an object other than *r.object*.

The following theorem gives the cumulative complexity of our on-line algorithm in the piecewise linear model.

**Theorem 5.6.** Assume that each object can change its velocity at most $m$ times. Then the cumulative complexity of the on-line algorithm for the uncertain case is $O(n^2m\alpha(n)\log n)$.

**Proof.** There are four types of processing involved in the on-line algorithm, i.e., the processing of velocity-vector updates, the processing of intersection events, the processing of max 1-level events, and the processing of implicit updates to the object heap.

**Processing velocity-vector updates.** Since processing each velocity-vector update takes $O(n)$ time, the cumulative complexity for processing velocity-vector updates is $O(n^2m)$.

**Processing intersection events.** According to Theorem 5.2, there are $O(nm\alpha(n))$ parabola pieces in the max 1-level. Each min-curve has $O(m)$ parabola pieces. According to Lemma 5.1, there are $O(nm\alpha(n))$ intersections between each min-curve and the max 1-level. Thus there are totally $O(n^2m\alpha(n))$ intersections between $n$ min-curves and the max 1-level. Processing each intersection event takes $O(\log n)$ time to
remove from the event queue. Processing each intersection event takes $O(\log n)$ time. Thus the cumulative complexity of processing intersection events is $O(n^2m\alpha(n)\log n)$.

**Processing max 1-level events.** Each implicit update to the object heap triggers a max 1-level event. According to the analysis given in §5.4.2.4, there are $O(nm\alpha(n)\log n)$ implicit updates to the object heap and thus there are $O(nm\alpha(n)\log n)$ max 1-level events. Processing each max 1-level event takes $O(n)$ time. Thus the cumulative complexity of processing max 1-level events is $O(n^2m\alpha(n)\log n)$.

**Processing implicit updates to the object heap.** The cumulative complexity for processing implicit updates is $O(nm\alpha(n)\log^2 n)$ as discussed in §5.4.2.4.

Thus the cumulative complexity of our on-line algorithm in the piecewise linear model is $O(n^2m\alpha(n)\log n)$.

This is a good result in the following sense. According to Theorem 5.5, the number of answer-pairs is $\Omega(n^2m)$, which is a lower bound for the cost of any algorithm for maintaining the P1NN set. Our on-line algorithm is only $\alpha(n)\log n$ higher than this lower bound.

### 5.5.2.3 Extension to $k>1$

For $k>1$, we want to use MOH to monitor the max $k$-level object. For this purpose, we modify MOH as follows. Each internal node $x$ in MOH stores three items $x.kth$, $x.set$, and $x.time$, such that $x.kth$ is the object that has the $k$-th smallest maximum possible distance up to time $x.time \triangleq 1$ among all the objects in $ObjectSet(x)$ and $x.set$ is the set of the $k$ objects that have the smallest maximum possible distances up to time $x.time \triangleq 1$ among all the objects in $ObjectSet(x)$. The purpose of storing $x.set$ is to enable the recomputation of $x.kth$ when an implicit update occurs.

It is not difficult to see that, when $k$ is a constant, the complexity of our on-line algorithm for $k>1$ is the same as that for $k=1$. 
5.6 Evaluation by experiments

In this section we evaluate our work by experiments using real-world data. In the evaluation we compare object heap with two existing algorithms namely kinetic tournament [91] and off-line processing [89]. We also compare the certain case with the uncertain case. The analysis conducted in previous sections compared these algorithms/cases in terms of the asymptotic worst-case complexity. Experiments help us understand in practice how they compare with each other. The performance measure is the cumulative processing time in a course during which the database receives a sequence of updates. All the compared algorithms are implemented in Java and executed on a Dell PowerEdge C6145 server with 48GB memory, 16-core AMD Opteron Magny-Cours 6136@2.4GHz. The operating system is Red Hat Enterprise Linux 6.

5.6.1 Test Data

The test data was provided by Shanghai Jiaotong University. The data contains GPS traces collected from over 4,000 taxis running in the Shanghai urban area for 28 days [112]. Each GPS trace is a sequence of GPS points where each GPS point is a triple \((x, y, t)\) which indicates that a taxi is at location \((x, y)\) at time \(t\). The difference in time between two consecutive GPS points is 1 minute. 4,000 taxis are too few for evaluation of scalability. In order to create a large-scale database, we split the GPS trace of each taxi into segments so that the time length of each segment is one hour. Then for each 1-hour segment we created a moving object to travel it. Furthermore, all the moving objects were temporally aligned so that they start to move at the same time (time 0). The rationale for this treatment is that in reality the movement of a taxi is more or less like a random walk in the city. By arbitrarily splitting a GPS trace into segments and letting them start at the same time, we put pieces of a random walk together, which preserves the characteristics of random walk. For each of the 28 days and each taxi, we created moving objects using the 9-hour period from 8am to 5pm. In this way we created a pool of 1,087,893 moving objects where each object moves for 3600 seconds starting from time 0.
Each GPS point is treated as a velocity-vector update. A taxi is assumed to move linearly at a constant speed between two consecutive GPS points. On average each moving object generates one velocity-vector update per minute. For the uncertain case the uncertainty region radius is a parameter which ranges from 1 to 30 meters. Indeed, the accuracy of a typical GPS receiver nowadays is <15 meters for 95% of time (see e.g., [113]). With GPS augmentation technologies such as EGNOS and WAAS which are available in market, the positioning accuracy can be improved to <3 meters for 95% of time (see [113, 114]). For an experiment run, $N$ objects are randomly picked up from the pool. $N$ is a system parameter ranging from 100 to 1,000,000. A random object is selected as the query object.

All the system parameters and their values are listed in Table XII.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Unit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Objects</td>
<td>$N$</td>
<td></td>
<td>100, 1000, 10000, 100000, 1000000</td>
</tr>
<tr>
<td>$k$</td>
<td></td>
<td></td>
<td>1 to 50 with increment of 5</td>
</tr>
<tr>
<td>Uncertainty region radius (for uncertain case)</td>
<td>$r$</td>
<td>meter</td>
<td>1, 5, 10, 15, 20, 25, 30</td>
</tr>
<tr>
<td>Simulated time</td>
<td></td>
<td>second</td>
<td>3600</td>
</tr>
</tbody>
</table>
5.6.2 Experimental results

In §5.6.2.1 we compare object heap and kinetic tournament. In §5.6.2.2 we compare on-line processing (object heap) and off-line processing. In §5.6.2.3 we compare the certain case and the uncertain case. In §5.6.2.4 we examine the space overhead of query processing.

5.6.2.1 Comparison between Object Heap and Kinetic Tournament

Figure 55 shows the query processing time of object heap and kinetic tournament as a function of the number of objects when $k=1$. Figure 56 shows the query processing time of the two algorithms as a function of $k$ when $N=10000$. Here the query processing time includes the time for initialization of the data structures, for processing of implicit updates as well as velocity-vector updates. From the two figures it can be seen that the query processing time is almost the same for object heap and kinetic tournament. The comparison for other parameter configurations point to the same conclusion. On the other hand, the worst case complexity of kinetic tournament is higher than that of object heap by a factor of $\log(N)$ (see §5.4.2). The reason for the difference between the experimental results and the worst case analysis is as follows. As will be demonstrated in §5.6.2.3, in both methods, the cost of query processing is dominated by the cost for processing velocity-vector updates. In both methods, processing a velocity-vector update involves percolating in the respective data structures. It has been established in [115] that the average height of percolation is independent of $N$ for an insertion to a binary heap even though in the worst case the height of percolation is $\log N$. This is also true for kinetic tournament and object heap, as shown in Table XIII. The table gives the percolation heights, which dominate the cost of update processing for the two methods. Thus the factor of $\log(N)$ in the worst case does not matter much in the average case.
Table XIII. Cost components when processing a velocity-vector update

<table>
<thead>
<tr>
<th>Number of objects (N)</th>
<th>100</th>
<th>1000</th>
<th>10000</th>
<th>100000</th>
<th>1000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kinetic Tournament</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>average height of percolation in tournament tree</td>
<td>1.9</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>1.9</td>
</tr>
<tr>
<td>average height of priority queue percolation-up</td>
<td>1.5</td>
<td>1.5</td>
<td>1.4</td>
<td>1.4</td>
<td>1.4</td>
</tr>
<tr>
<td>average height of priority queue percolation-down</td>
<td>0.9</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Object Heap</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>average height of percolation</td>
<td>3.2</td>
<td>3.3</td>
<td>3.3</td>
<td>3.3</td>
<td>3.3</td>
</tr>
</tbody>
</table>

Figure 55. Query processing time versus number of objects
5.6.2.2 Comparison between On-line Processing and Off-line Processing

First let us consider the scenario in which there are no velocity-vector updates. In this case off-line processing and on-line processing compute the same number of answer-pairs. In order to create this scenario we used only the first two GPS points of each moving object. We executed off-line processing and on-line processing to compute the answer-pairs for the period of time starting from 0 until infinity. For off-line processing we used the divide-and-conquer algorithm introduced in [89]. For on-line processing we used object heap. The processing time of the off-line algorithm includes the time for one-time execution of the algorithm at time 0. The processing time of the on-line algorithm includes the time for initialization of the object heap and processing of implicit updates. Figure 57 shows the comparison result in the certain case. From the figure it can be seen that the cost of on-line processing is higher than that of off-line processing by two orders of magnitude. However the absolute difference remains small (up to 10 seconds). Indeed, when there are no updates there is no advantage of doing on-line processing.

Now we consider the case in which there are velocity-vector updates. For both off-line processing and on-line processing, we assumed that an object moves linearly to infinity after its last velocity-vector
update. In this case, when a velocity-vector update occurs, off-line processing is executed to re-compute the answer-pairs for the period of time starting from current time until infinity. Thus, the processing time of the off-line algorithm includes the cumulative time for executions of the algorithm upon each velocity-vector update. The processing time of the on-line algorithm includes the time for initialization of the object heap, processing of implicit updates as well as velocity-vector updates (i.e., explicit updates).

Figure 58 shows the comparison as a function of the total number of velocity-vector updates for all the objects in the certain case. From the figure it can be seen that when there are fewer than 30 updates, off-line is better than on-line. However, when there are many updates, on-line is much better than off-line. When the number of updates reaches 100,000, the cost of off-line is one thousand times higher than that of on-line. Intuitively, if there are many updates, then off-line processing totally computes more answer-pairs than on-line processing and thus costs more time.

Figure 59 shows the comparison in the uncertain case. On-line processing starts to outperform off-line processing when the number of updates is higher than 100. Notice that there are totally 600,000 updates for all the objects when N=10000 whereas the maximum number of updates in Figures 58 and 59 is 100,000. Thus on-line is clearly much better than offline for practical usage, as the trend in Figures 58 and 59 shows.

![Figure 57. Comparison between on-line and off-line as a function of the number of objects when there are no updates in the certain case](image-url)
Figure 58. Comparison between on-line and off-line as a function of the total number of velocity-vector updates for all the objects in the certain case

Figure 59. Comparison between on-line and off-line as a function of the total number of velocity-vector updates for all the object in the uncertain case
5.6.2.3 Comparison between Certain Case and Uncertain Case

In this subsection we compare the certain case and the uncertain case in terms of the number of answer-pairs, the number of objects returned by each answer-pair, and the query processing time.

Number of Answer-pairs. Figures 60 to 62 show the results for the number of answer-pairs. The data point for \( N=1,000,000 \) for the uncertain case in Figure 60 is not attainable due to a prohibitively long computation time. From the figures it can be seen that the uncertain case has much more answer-pairs than the certain case. Furthermore, the difference between the two cases increases with the number of objects, \( k \), and the uncertainty region radius. The huge difference between the certain case and the uncertain case matches the analytical result (recall the \( O(n) \) factor of difference according to Theorem 5.2 and Theorem 5.4).

Size of \( P_kNN \) Set. Now we study the size of the \( P_kNN \) set which is the number of objects included in an answer-pair in the uncertain case. Figures 63 to 65 show the size of the \( P_kNN \) set as a function of the number of objects, \( k \), and the uncertainty region radius, respectively. From these figures it can be seen that the size of the \( P_kNN \) set is bigger than \( k \). Furthermore, this size increases with the number of objects, \( k \), and the uncertainty region radius. Figure 64 compares the average size of an answer set returned by the uncertain case and that by the certain case. The figure shows that the uncertain case returns about 12 more objects than the certain case when there are 10000 objects and the uncertainty region size is 15 meters. It is interesting that the average number of extra objects returned by the uncertain case is independent of \( k \).
Figure 60. Number of answer-pairs versus number of objects

Figure 61. Number of answer-pairs versus $k$
Figure 62. Number of answer-pairs versus uncertainty region radius

Figure 63. Size of $P_k$NN set versus number of objects
Query Processing Time. We focused on on-line processing. For on-line processing, every GPS point was treated as a velocity-vector update. The query processing stopped at the 3600th simulated second which is the end time of the GPS trace for all moving objects. Observe that the processing time may end before or after the simulated time. For example, if the average time it takes to process an update is bigger
than the average inter-arrival time for updates, then the query processing time will be larger than the simulated time, otherwise it will be smaller. Figures 66 to 68 show the results for the query processing time. The data point for $N=1,000,000$ for the uncertain case in Figure 66 is not attainable due to a prohibitively long computation time. On the other hand, on-line processing easily scales to $N=1,000,000$.

Furthermore, the difference between the two cases increases with the number of objects. Particularly, the processing time of the uncertain case grows much faster than that of the certain case as the number of objects grows (see Figure 66). This phenomenon matches the analytical results (recall the $O(n)$ factor of difference according to Table XI).

One fact that is a bit surprising is that the query processing time of the uncertain case does not change with the uncertainty region radius (see Figure 68). This is surprising because Figure 62 shows that the number of answer-pairs increases with the uncertainty region radius and thus presumably the query processing time should also increase with the uncertainty region radius. The reason for the phenomena shown in Figure 68 is as follows. Recall the proof of Theorem 5.6 in §5.5.2.2. The cost of query processing for the uncertain case is dominated by three components: processing intersection events, processing max 1-level events, and processing implicit updates to the object heap. Observe that when all the other parameters are fixed and only the uncertainty region radius changes, the cost of processing max 1-level events and processing implicit updates to the object heap are fixed. The cost of processing max 1-level events is fixed because the algorithm always computes the critical intersections between the max 1-level and the min-curves of all the other $N−2$ objects. The cost of processing implicit updates to the object heap is fixed because the change of the max 1-level is not affected by the uncertainty region radius. The only component that is affected by the uncertainty region radius is processing intersection events. However, the cost of this component is negligible as shown in Table XIV. Table XIV also shows that the most time-consuming part is processing max 1-level events. In this part, time is consumed for computing intersections between the max 1-level and the min-curves.
Table XIV. Processing time distribution for the uncertain case when $N=10000$, $k=1$, and $r=15$m.

<table>
<thead>
<tr>
<th>Processing component</th>
<th>Processing time (second)</th>
<th>Processing intersection events</th>
<th>Processing max 1-level events</th>
<th>Processing implicit updates to object heap</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0.1</td>
<td>621</td>
<td>0.2</td>
</tr>
</tbody>
</table>

Table XV shows the processing time distribution for the certain case when $N=1,000,000$ and $k=1$. From the figure it can be seen that most of the query processing time is spent on handling velocity-vector updates. This is true for other values of $N$ and $k$ that we tested.

Table XV. Processing time distribution for the certain case when $N=1000000$ and $k=1$.

<table>
<thead>
<tr>
<th>Processing component</th>
<th>Initialization</th>
<th>Processing implicit updates</th>
<th>Processing velocity-vector updates</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4</td>
<td>15</td>
<td>190</td>
</tr>
</tbody>
</table>

Finally, observe that the query processing time of the uncertain case reaches 10000 seconds when there are 30,000 objects (see Figure 66). On the other hand, the simulated time is only 3600 seconds. In this case, the time that is needed to process an update exceeds the time length between two consecutive updates. Query processing becomes infeasible. Clearly, for any query processing algorithm, there exist an update rate and a problem size beyond which the algorithm falls behind.
Figure 66. Query processing time versus number of objects

Figure 67. Query processing time versus $k$
5.6.2.4 Space Overhead

We focus on the space overhead of on-line processing. First let us consider the space overhead in the certain case. The space overhead of object heap comes from the binary tree. Figure 69 shows the space

Figure 68. Query processing time versus uncertainty region radius

Figure 69. Space overhead of object heap versus number of objects
overhead of object heap as a function of the number of objects for $k=1$ and $k=50$ respectively. From the figure it can be seen that the space overhead is linear in the number of objects. This is because the number of nodes in the binary tree is linear in the number of objects. Observe that when there are 1 million objects, the space overhead reaches 800MB when $k=1$. This suggests that object heap is well suitable for in-memory processing. The space overhead can be reduced by storing the binary tree in an array and thus getting rid of pointers. However, space optimization is not the focus of this work.

Let us examine the space overhead for different values of $k$. One would think that the space overhead should increase with $k$ since the size of the $k$NN set stored at each node of the binary tree increases with $k$ (see §5.4.2.5). However, Figure 69 shows that the space overhead for $k=50$ is actually smaller than that for $k=1$. The reason for this is as follows. Notice that each leaf node in the binary tree stores at most $k$ objects. It is not difficult to see that the memory overhead of the binary tree is

$$
\left(2 \cdot \left\lceil \frac{N}{k} \right\rceil - 1 \right) (H + k \cdot c) \approx 2 \cdot N \cdot \left(\frac{H}{k} + c\right) \quad (1)
$$

where $H$ is the fixed space overhead of each node excluding the overhead of the $k$NN set, $c$ is the space overhead for each member of the $k$NN set. From Equation (1) it can be seen that the space overhead decreases as $k$ increases.

Now we study the space overhead in the uncertain case. Compared with the certain case, the only additional data structure used by the uncertain case is the event queue (see §5.5.2.1). We traced the average length of the event queue and the results are shown in Table XVI. From Table XVI it can be seen that the additional space overhead introduced by the uncertain case is negligible.

Table XVI. Average length of the event queue in the uncertain case when $k=1$ and $r=15m$.

<table>
<thead>
<tr>
<th>Number of objects ($N$)</th>
<th>100</th>
<th>1000</th>
<th>10000</th>
<th>100000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average length of event queue</td>
<td>1.0</td>
<td>1.0</td>
<td>1.2</td>
<td>1.3</td>
</tr>
</tbody>
</table>
5.7 Summary

The theoretical results developed in this chapter are summarized in Table XI in §5.1. Furthermore, we evaluated the proposed algorithms by experiments using real-world GPS traces. The results showed that (i) even though object heap is better than kinetic tournament by a factor of \( \log(n) \) in the worst case, it is equally efficient as the latter one for the data set we tested, in the average case; (ii) without any indexing structure, query processing easily scales to 1 million objects in the certain case and it scales to 10,000 objects in the uncertain case; (iii) on-line processing is much more efficient than off-line processing for practical update rates; and (iv) the space overhead of query processing fits in the main memory.
6 TRANSPORTATION MODE DETECTION

The transportation mode such as walking, cycling or on a train denotes an important characteristic of the mobile user’s context. In this chapter we present an approach to inferring a user’s mode of transportation based on the GPS sensor on her mobile device and knowledge of the underlying transportation network. The transportation network information considered includes real time bus locations, spatial rail and spatial bus stop information. This information can be disseminated by mobile peers via peer-to-peer communication.

In section 6.1 we summarize our contributions. At the end of this section we outline the structure of this chapter.

6.1 Contributions

In ubiquitous and context aware computing, understanding the mobility of a client from sensor data is an important area of research. The transportation mode, such as walking, cycling, or train denotes some characteristics of the mobile user’s context. With knowledge of a traveler’s transportation mode, targeted and customized advertisements may be sent to the traveler’s device. For example, if we discover that Alice is driving by car, the system may send her gas coupons or vehicle service specials.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Another motivation for transportation mode detection is transportation surveys. Travel demand surveys have taken multiple formats, such as telephone interviews and questionnaires. These data collection strategies rely on manual labeling of data after the trip, and thus, inaccuracies are introduced. For example, a traveler may not recall the exact time that she/he boarded a transportation mode. Using
GPS devices is more reliable for reporting accurate location, trip time, and trip duration [116, 117, 118]. Hence, if the precise transportation modes of individual users are recognized, it is possible to provide a more realistic travel demand picture.

Many GPS trace sharing social networks have been implemented [119-122]. These social networks enable friends to upload and share their GPS traces. Knowledge of transportation mode, added to these GPS traces, will enable the users to reflect on their past motion more meaningfully. It also allows users to obtain additional information from their friends’ travel experience. Additionally, awareness of transportation mode of a user may help to determine the user’s carbon footprint, or track the amount of calories burnt. Another application of transportation mode detection is crowd-sourced real-time traffic information in which traffic speeds are aggregated from probes such as mobile phones carried by travelers. Transportation mode detection enables the aggregation system to filter out the speed data submitted by non-motorized travelers or travelers on trains.

Transportation mode detection has been documented in the literature [123-127]. The existing approaches share the following general principle. First, from historical data, build a classification model in terms of mobility patterns. Then, when the transportation mode is to be determined, collect input from mobility sensors and feed the input to the classification model. The state-of-the-art is the technology developed in [125] which fuses input from GPS receiver and accelerometer. However, [125] only distinguishes between walking, running, biking, and motorized transport. It (i.e. [125]) does not distinguish between various modes under motorized transport, such as driving versus taking a bus. As shown in [125], using only GPS information reduces detection accuracy, compared to using both GPS and accelerometer information. Clearly, the accuracy of transportation mode detection may be higher if one utilizes more sensors. However, the objective of our work is to determine the added value of the transportation network data. Specifically, we consider adding to GPS data the real time locations of buses, spatial polylines representing rail line routes, and bus stop locations.
In this chapter we propose a method that is able to distinguish not only between non-motorized transport and motorized transport, but also between various motorized modes including automobile, bus, and aboveground train. Additionally, if we determine that the transportation mode is bus, we further provide information on which particular bus the client is travelling on.

We follow the general principle of sensor data fusion and classification that has been used in prior work [123-128]. Fusing GPS sensor data with external transportation network data makes transportation mode detection more robust. Intuitively, different transportation modes have different mobility patterns. For example, motorized transport generally has a higher speed than non-motorized transport. For another example, being constrained by a road, people driving a car or taking a bus cannot change their heading direction as flexibly as if they are walking or cycling. On the other hand, relying on a single type of input does not always work. For example, movement at 7 km/hr may be a brisk walk, or a slowly moving car or bus, in congestion.

Distinguishing between motorized and non-motorized transportation mode is not a difficult problem. However, with multiple motorized transportation modes, the problem becomes more difficult since buses, cars and trains may have similar GPS or accelerometer readings. We show that using a transportation network with real time and static spatial data, we can obtain high detection accuracy for various motorized and non-motorized transportation modes.

In summary, our approach is the first to address transportation mode detection using external transportation network data such as real time bus locations; this is in addition to mobile device sensor-information used in traditional approaches to the problem. Our contributions are as follows: (1) In addition to the traditional features on average speed and average acceleration, we identify for the first time the features of average bus closeness, average rail closeness, and average candidate bus closeness as the most effective features related to transportation mode detection, (2) The proposed work is the first to distinguish between motorized modes (bus, car, train) with such high accuracy, (3) There are other works that distinguish between cars and buses [123, 124, 126, 127]; however, the proposed approach is the first
to consider aboveground train as a transportation mode, (4) We introduce a zip-code based indexing and pruning technique to speed up the feature computation, and (5) We present simulation results and real world results, showing the efficiency of the proposed approach.

The rest of this chapter is organized as follows. In §6.2 we discuss the relevant work. In §6.3 we introduce the data model and the general idea of our mode detection algorithm. In §6.4 we describe the system architecture and introduce the transportation network data. In §6.5 we present the selection of mode detection features. In §6.6 we evaluate our algorithm using real data. In §6.7 we conclude the chapter.

6.2 Relevant Work

The work of Zheng et al. [124, 126] is based on transportation mode detection from GPS data alone; the authors introduce a robust and novel set of machine learning features that are sensitive to certain traffic and weather scenarios. Our work is different in that we consider transportation network data such as the real time location of buses to build classification features. Additionally, [124, 126] do not consider train as a transport mode. The approach proposed in this chapter is over 17 % more accurate than [124, 126].

In [123, 127], the authors use an unsupervised learning technique to detect the transportation mode of a traveler. The transportation modes that are detected in [123, 127] include buses, cars and walk. The work in [123, 127] is able to predict the traveler’s goals, such as trip destination and trip purpose. In addition to GPS and GIS data, [123, 127] use historical information about the user. Historical information includes, past user trips and information about where the users parked their cars. In our approach we do not consider historical information about the user. Furthermore, we use a supervised learning mechanism to detect transportation modes from the set \{WALK, BUS, DRIVING, TRAIN, STATIONARY, BIKE\}. Another difference is that we use different transportation network data than [123, 127] do. Particularly we use real time bus locations, rail line spatial data, and bus stop spatial data. [123, 127] use historical information about the bus stops at which a user boards, and where the user parks her/his vehicle.
Importantly, the proposed bus stop feature is different than that in [123, 127]; the proposed classification feature captures the number of bus stops and duration at bus stops. A weakness of [123, 127] is that the users’ motion pattern such as where the user parks her/his vehicle daily are taken into consideration, and therefore the model relies on background information about the user. The accuracy of the proposed approach is higher than that of [123, 127] by 9%.

The proposed approach uses a single sensor (i.e. GPS) on the mobile device. There have been studies that consider multiple sensors for transportation mode recognition [125, 129, 130, 131, 132]. In [129, 130], over 20 sensors that are wearable on the human body are used. The input to the classification model includes information on the user’s body condition such as temperature, heart rate and GPS position. We consider a smaller number of sensors, but add transportation network data. We believe that it is unlikely for normal users to carry over 20 sensors daily. [131] uses multiple accelerometers and [132] uses a single sensing unit with multiple sensors (accelerometer, audio, and barometer) for activity detection. The state of the art is [125] which uses GPS and accelerometer sensors for transportation mode detection. However, [125] does not distinguish between motorized transportation modes such as car and bus. This limitation is due to the similarity in features of these two modes of transportation, especially in traffic or extreme weather. Using GPS and GIS data, as shown in the proposed approach, can achieve a very high detection accuracy, as in [125]. However, in the proposed approach we distinguish between motorized transportation modes and we do not use accelerometer as in [125]. Figure 70 summarizes the related works that uses GPS.

The work in [133] is purely based on GSM, whereas we use GPS. In [134], the only sensor considered is the triaxial accelerometer. In [118], the authors’ objective is to conserve mobile devices resources such as battery life. Thus, in [118] only critical location points are submitted. Furthermore, the set of classification features used in our work is different from [118].

Our prior research in [128] has a different focus; it considers extracting the semantic location from outdoor positioning systems. Likewise, [135] learns and recognizes the places a mobile user visited by observing the Wi-Fi and GSM radio fingerprints. This work does not consider Wi-Fi or GSM information.
Instead, we consider GPS and transportation network data. Transportation network data is available freely to the public in many cities [136, 137, 138].

<table>
<thead>
<tr>
<th>Classes</th>
<th>Sensor</th>
<th>Duration of test data</th>
<th>Users</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>[124] driving, bus, bike, walk</td>
<td>GPS</td>
<td>10 months</td>
<td>65</td>
<td>76.2%</td>
</tr>
<tr>
<td>[125] still, walk, run, bike, motor</td>
<td>GPS, accelerometer</td>
<td>50 days</td>
<td>16</td>
<td>93.6%</td>
</tr>
<tr>
<td>[123,127] walk, bus, car</td>
<td>GPS, GIS</td>
<td>60 days</td>
<td>1</td>
<td>84%</td>
</tr>
<tr>
<td>[126] car, bus, bike, walk</td>
<td>GPS</td>
<td>6 months</td>
<td>45</td>
<td>74%</td>
</tr>
</tbody>
</table>

Figure 70: Related work with GPS sensor

6.3 Preliminaries

In this section, we discuss the data model and the general idea of our algorithm.

6.3.1 Data Model

Definition 1. GPS sensor report. A sensor GPS report \( p_i \) represents data submitted from the GPS sensor embedded on a traveler’s mobile device. The format of the report is \(<\text{lat}, \text{lon}, t, v, h, acc>\) where: 
\( \text{lat} \) represents the latitude; 
\( \text{lon} \) represents longitude; 
\( t \) represents the timestamp of the sensor report; 
\( v \) represents the current ground speed of the device; 
\( h \) represents the direction of travel; and 
\( acc \) represents the accuracy level of the latitude and longitude coordinates.

The measurement units of the GPS sensor report attributes are: 
latitude (\( \text{lat} \)) and longitude (\( \text{lon} \)) are in decimal degree; 
current ground speed (\( v \)) is measured in meters per second; 
direction of travel (\( h \)) is specified in degrees counting clockwise from true north; 
accuracy level (\( acc \)) is defined in meters; and 
time \( t \) is in seconds.
Definition 2. A GPS trace $T$ is a sequence of GPS sensor reports, $T = p_0 \rightarrow p_1 \rightarrow \cdots \rightarrow p_k$, where the timestamps in the sequence strictly increase.

6.3.2 General Idea

In general, our algorithm is a supervised learning mechanism with two stages. In stage 1 (learning stage), the data from the GPS sensor report is merged with the transportation network data and labeled ground truth. This data is used to create a classification feature set that we use to train our classification model. In this stage, mobile devices submit GPS sensor reports every $t$ seconds, where $t$ is a system parameter. These incoming sensor reports are labeled with the corresponding transportation modes.

Then, in stage 2 (inference stage), to determine a traveler’s transportation mode, we first extract the same classification features as in stage 1. Subsequently, given the features, the classification system predicts the transportation mode of the traveler in a probabilistic format.

Specifically, our mode detection algorithm fuses inputs from the mobile devices’ GPS receivers with real time locations of buses, rail line and bus stop location data. GPS technology is a built-in feature of many mobile devices, such as IPhones, BlackBerrys and Android phones. Given a GPS trace of a traveler, one way to build the classification model is as follows. For each GPS sensor report in the trace, various features including the closest Euclidian distance to rail lines, closest Euclidian distance to buses and closest Euclidian distance to bus stops are computed. Mean speed, heading, and acceleration are also obtained over a time window. These features form a sensor feature vector. The feature vector, plus the transportation mode label of the associated time interval, forms a training example. In this way, a training set is constructed. This procedure is illustrated by Figure 71.

GPS location, heading, speed and acceleration are features that have been used in existing studies. However, features like closest Euclidian distance to rail line, closest Euclidian distance to buses and bus stop closeness rate have never been used before. These are newly introduced in this chapter.
6.4 Transportation Mode Detection

In this section, we present the system architecture and introduce the transportation network data that is utilized in our mode detection algorithm.

6.4.1 System Architecture

We use a centralized system architecture. Each mobile device submits its GPS sensor reports to the central server. After the central server receives a time window amount of GPS sensor reports, it predicts the transportation mode used, and sends this prediction to the mobile device. We believe this centralized system is more platform independent than the distributed counterparts, where classification is done directly on the mobile device. Furthermore, since the mode detection is performed at the central server, there is no need to store transportation network data on the mobile device. Hence, the centralized model consumes less of the device’s memory, less processing time, less bandwidth, and less battery power. On the other hand, the distributed model is location privacy aware, since the location of the user is not submitted to a central authority. The privacy issue with the centralized system is addressed in [139, 140].

6.4.2 Transportation Network Data

We fuse data from the GPS sensor reports with data from the transportation network to create the classification feature vector. Specifically, for the city of Chicago, Illinois, USA, we consider: (1) real time location of public passenger buses, (2) rail line spatial information, and (3) public passenger bus stop

Figure 71. Generating classification examples from GPS sensor and transportation network data
spatial data. In §6.5, we will discuss the procedure to create the classification features from GPS sensor reports and transportation network information.

6.4.2.1 Real time bus location

Real time locations of public passenger buses for the city of Chicago is available to the public [137]. Each of these buses has a GPS receiver and can determine its location, and then report the location back to some server. Likewise, real time public transit tracking is performed in many other cities such as London, New York, San Francisco, Toronto, and Washington. For the city of Chicago, the system considers the real time locations of buses belonging to Chicago Transit Authority (CTA). CTA has over 1,700 buses in service that operates over 140 routes. On an average weekday, 1.7 million rides are taken [136]. The real time locations of the buses are updated every 20-30 seconds, and the data is available freely to the public as an API in XML format. Information available about the CTA buses includes: route, latitude, longitude, final stop, bus identification, and direction. In Figure 72, we create a “MashUp” using Google Maps [141] and the real time locations of the CTA buses.

6.4.2.2 Rail Lines

Spatial data of the rail tracks (train routes) in the city of Chicago is also available to the public [138], as geometric polylines. Figure 73 depicts a diagram of the Chicago Transit Authority (CTA) rail network. This rail line trajectory’s location information is used in the proposed system. Spatial rail line information fused with GPS sensor data creates a classification feature in the proposed system. The classification feature is the Euclidian distance between the traveler’s mobile device and closest rail line.

6.4.2.3 Bus Stops

The CTA services over 11,577 bus stops [136]. Spatial information, name, and identification of these public passenger bus stops are available [136]. Public passenger bus stops’ spatial information, merged with GPS sensor data from the traveler’s device, creates a classification feature in the proposed system.
Figure 72: Real-time bus locations
6.5 Classification Feature Selection

This section deliberates the classification features used in the proposed transportation mode classification system. Additionally, the motivation for each feature, and the algorithm used for calculating the feature values are discussed. In this chapter we explore 4 novel classification features related to motorized transportations: (1) average bus location closeness, (2) candidate bus location closeness, (3) average rail line trajectory closeness, and (4) bus stop closeness rate. This is in addition to the traditional features considered in the literature: average accuracy of GPS coordinates, average speed, average heading change, and average acceleration. In the rest of this section we describe each of the novel features and the traditional features.

6.5.1 Average accuracy of GPS coordinates

The estimated horizontal accuracy is a measure of the confidence on the location reported by the GPS sensor; it is a component of a GPS sensor report (see Definition 1). The accuracy is reported in meters. Different transportation modes should have different estimated accuracies. For example, traveling by aboveground trains should have worse accuracy than walking, since walking has a clearer view of the GPS satellites in the sky. Additionally, we consider the average accuracy for a set of GPS reports as a feature, instead of the instantaneous accuracy as done in [118]. Taking the average accuracy is more realistic since the GPS system may introduce uncertainties.

Let \( \{ p_1, p_2, p_3, \ldots, p_n \} \) be a finite set of GPS sensor reports submitted from the traveler’s mobile device within a time window.

\[
\text{Average accuracy} = \left( \frac{\sum_{i=1}^{n} p_i^{\text{acc}}}{n} \right)
\]

(1)

where \( p_i^{\text{acc}} \) is the estimated accuracy of the reported GPS position.
6.5.2  **Average speed**

In terms of speed, we use the speed value returned by the GPS sensor when it is available; this is more accurate than calculating the speed from consecutive GPS location points [125]. Otherwise, if the direct speed is not available, it can be computed from consecutive location changes. For a sequence of GPS reports we compute the average speed. This feature has been used in many existing works [124, 125, 130].

Let \( \{p_1, p_2, p_3, p_4, \ldots, p_n\} \) be a finite set of GPS sensor reports submitted within a time window.

\[
\text{Average speed} = \frac{\sum_{i=1}^{n} p_i}{n} \quad (2)
\]

where \( p_i \) is the current ground speed obtained from the GPS sensor report.

6.5.3  **Average heading change**

The heading is the direction from true north. For a set of GPS reports, we compute the average heading change. The heading change is an important feature for distinguishing between motorized and non-motorized transportation mode as observed by Zheng et al. [124]. This proposed classification feature is different from the heading feature in [124] because we compute the average heading change whereas [124] computes the heading change rate. The heading change rate in [124] is defined to be the number of times the heading change exceeds a certain threshold. It is computed as the ratio \(|P_c| / \text{distance}\), where \(|P_c|\) represents the number of points where the traveler changes heading beyond the heading threshold. The heading change rate as defined in [124] cannot be used to distinguish between transportation modes with heading change rate below the chosen heading threshold. Let \( \{p_1, p_2, p_3, p_4, \ldots, p_n\} \) represent a finite set of GPS reports submitted within a time window.

\[
\text{Average heading change} = \frac{\sum_{i=1}^{n} |p_i^h - p_{i-1}^h|}{n} \quad (3)
\]

\( \forall 2 \leq i \leq n \)

where \( p_i^h \) is the direction from true north included in the GPS sensor report.
6.5.4 **Average acceleration**

Let \( \{p_1, p_2, p_3, \ldots, p_n\} \) be the finite set of GPS reports submitted within a time window.

\[
p_i^{\text{acceleration}} = \frac{(p_i^v - p_{i-1}^v)(p_i^t - p_{i-1}^t)}{\forall 2 \leq i \leq n}
\]

Average acceleration = \( \frac{\sum_{i=1}^{n} p_i^{\text{acceleration}}}{n} \)  

where \( p_i^{\text{acceleration}} \) is the acceleration of the mobile device.

6.5.5 **Bus location closeness**

This feature aggregates the traveler’s GPS location with the real time locations of public passenger buses. Bus location closeness is useful for determining if the mobile device is on a bus or not. We develop two algorithms to determine if a mobile user is traveling by bus; (1) **average bus closeness**, and (2) **candidate bus closeness**.

Let the location of the mobile user at time \( t \) be represented as \( p_t^{\text{loc}} \), based on the GPS sensor report. Also, let the \( m \) buses in the city be \( \text{bus}_1 \) to \( \text{bus}_m \), where \( \text{bus}_x^{\text{loc}} \) is the location of bus \( \text{bus}_x \) at time \( t \). Below, line 1 shows the mobile user’s location trace. Line 2 to line 5 represent the location traces of all the \( m \) buses (\( \text{bus}_1 \) to \( \text{bus}_m \)).

\[
p_1^{\text{loc}}, p_2^{\text{loc}}, p_3^{\text{loc}}, p_4^{\text{loc}} \ldots p_n^{\text{loc}} \\
\text{bus}_1^{\text{loc}}, \text{bus}_2^{\text{loc}}, \text{bus}_3^{\text{loc}}, \text{bus}_4^{\text{loc}} \ldots \text{bus}_{1,n}^{\text{loc}} \\
\text{bus}_2^{\text{loc}}, \text{bus}_2^{\text{loc}}, \text{bus}_2^{\text{loc}}, \text{bus}_2^{\text{loc}} \ldots \text{bus}_{2,n}^{\text{loc}} \\
\text{bus}_3^{\text{loc}}, \text{bus}_3^{\text{loc}}, \text{bus}_3^{\text{loc}}, \text{bus}_3^{\text{loc}} \ldots \text{bus}_{3,n}^{\text{loc}} \\
\ldots \\
\text{bus}_m^{\text{loc}}, \text{bus}_m^{\text{loc}}, \text{bus}_m^{\text{loc}}, \text{bus}_m^{\text{loc}} \ldots \text{bus}_{m,n}^{\text{loc}}
\]

**Average bus closeness (ABC)**

From GPS sensor reports \( \{p_1, p_2, p_3, \ldots, p_n\} \) that are submitted within a time window, we obtain the set of locations points \( \{p_1^{\text{loc}}, p_2^{\text{loc}}, p_3^{\text{loc}}, p_4^{\text{loc}} \ldots p_n^{\text{loc}}\} \). For each location point \( p_t^{\text{loc}} \), we compute \( d_t^{\text{bus}} \) as the Euclidian distance between \( p_t^{\text{loc}} \) and the closest bus \( \text{bus}_x^{\text{loc}} \) at time \( t \). Subsequently, given \( d_t^{\text{bus}} \), we
calculate the feature *average bus closeness* (ABC), as the average Euclidian distance of \((d_{1_{bus}}, d_{2_{bus}}, d_{3_{bus}},\ldots d_{n_{bus}})\), for the set of GPS sensor reports \(\{p_1, p_2, p_3, p_4,\ldots p_n\}\).

\[
ABC = \frac{\sum_{i=1}^{n} d_{i_{bus}}}{n} \quad (5)
\]

This feature is used to capture whether the traveler is traveling via bus transportation mode.

**Candidate bus closeness (CBC)**

First, we obtain the set of locations points \(\{p_{1_{loc}}, p_{2_{loc}}, p_{3_{loc}}, p_{4_{loc}}, \ldots p_{n_{loc}}\}\) from GPS sensor reports \(\{p_1, p_2, p_3, p_4,\ldots p_n\}\) that are submitted within a time window. For each location point \(p_{i_{loc}}\), we compute the Euclidian distance \(d_{j_{bus}}\) for \(1 \leq j \leq m\) to each bus \(bus_j\), in the set of all buses \(\{bus_1, bus_2, bus_3,\ldots bus_m\}\) at time \(t\).

Then, for each bus \(bus_j\), we compute the total Euclidian distance \(D_j\) over the time window as follows.

\[
D_j = \sum_{t=1}^{n} d_{j_{bus}} \quad 1 \leq j \leq m \quad (6)
\]

Given \(D_j\) for all the \(m\) buses, we compute CBC as follows.

\[
CBC = \min (D_j) \quad 1 \leq j \leq m \quad (7)
\]

The classification feature CBC is the minimum \(D_j\) value. Using the CBC feature requires more memory than the ABC counterpart, since the Euclidian distance from the device to every bus in the city needs to be computed and stored for each GPS sensor report. For ABC we only compute the distance to the closest bus. To the best of our knowledge, this work is the first to consider the real time location of buses for transportation mode detection.

**6.5.6 Rail line trajectory closeness**

This classification feature relates the traveler’s GPS location with spatial data representing the rail network. This feature may be useful to detect if a person is travelling on an aboveground train. For underground trains (subways), since GPS does not work well underground, this feature may not be effective. We do not consider subways in this work. The Euclidian distance \(d_{i_{rail}}\) between a person’s mobile device and the closest rail line is computed for each GPS sensor report \(p_i\). We then calculate the classification feature *average rail location closeness* (ARLC) as follows. Let \(\{p_1, p_2, p_3, p_4,\ldots p_n\}\) be a finite the set of GPS reports submitted within a time window.
\[ ARLC = \sum_{i=1}^{n} \frac{d_{\text{rail}}}{|n|} \quad (8) \]

To the best of our knowledge, the proposed work is the first to use this rail line feature for transportation mode detection. The predictive power of this feature on transportation mode detection is evaluated in §6.6.

6.5.7 **Bus stop closeness rate**

This classification feature relates the traveler’s GPS location with spatial bus stop data. First, from experiments we determine a *bus stop closeness threshold*. This threshold is a Euclidian distance measure and may be used to concur if a person is at a bus stop.

We calculate the classification feature BSCR (*bus stop closeness rate*) as follows. Let \( \{p_1, p_2, p_3, p_4, \ldots, p_n\} \) be a finite set of GPS sensor reports submitted within a time window. BSCR stands for the number of GPS sensor reports \( p_i \), whose Euclidian distance \( d_{\text{busstop}} \) to the closest bus stop, is less than the *bus stop closeness threshold* within a unit time.

\[ \text{BSCR} = \frac{|\text{PS}|}{\text{window size}} \quad (9) \]

where \( \text{PS} = \{p_i \mid p_i \in \{p_1, p_2, p_3, p_4, \ldots, p_n\}, d_{\text{busstop}} < \text{bus stop closeness threshold}\} \).

Experiments to determine bus stop closeness threshold value

Below, we explain how to obtain the *bus stop closeness threshold* value. For experiments, a traveler carried a mobile device and boarded a CTA bus. We then measured the Euclidian distance to the closest bus stop from the traveler’s device. From over 450 GPS sensor reports, we plot the graph of Figure 74. The vertical axis represents the Euclidian distance to the closest bus stop. The horizontal axis represents the corresponding GPS sensor report number. From Figure 74, we observe how the mobile device’s Euclidian distance to the closest bus stop fluctuates during the travel. When the traveler on a bus is at the bus stop, the distance is at a minimum. As the bus moves away from the bus stop, the distance increases. It peaks at the midpoint between two bus stops. Afterward, it decreases and reaches a new minimum when the bus reaches the next bus stop.
When the traveler is traveling via bus mode and at a bus stop, the Euclidian distance to the closest bus stop is less than 13 meters. Thus, for bus stop closeness threshold, we used a value of 13m.

For BSCR, we compute the number of times the Euclidian distance to the closest goes below the bus stop closeness threshold per unit time. We believe that if a traveler is traveling by bus, the BSCR should be greater than if they are not travelling by bus. We also evaluate the effectiveness of BSCR on predicting the transportation mode in the proposed work.

![Figure 74: Mobile user’ Euclidian distance to closest bus stop while riding a bus](image)

6.5.8 Zip code based Indexing and Pruning

Recall that from the user’s GPS sensor reports we compute the closest Euclidian distances from real time locations of buses, rail line trajectories, and bus stops. Doing a linear comparison with all these locations can be time consuming.

We build a flat indexing scheme using zip codes; this scheme alleviates the overhead of doing the costly linear comparisons. First, we pre-compute the zip codes for all the bus stops, bus routes and train lines. Then, for each zip code in the city, we maintain a bus stop candidate list, bus route candidate list, and rail line candidate list. We cache this zip code index on the central server. Next, when a mobile user submits a GPS sensor report, instead of doing a linear comparison with all the bus stops, buses, and rail
lines, we only compare against those in the same zip code from which the GPS sensor report was submitted.

In the proposed work, we compute the zip codes by reverse geocoding the spatial data. Reverse geocoding is the process of converting a location point to a readable address or place name. For example, reverse geocoding latitude: 41.976216 and longitude: -87.90331, produces the address 1-99 Access Road Chicago, Illinois, 60666, USA. From the address, we extract the zip code (i.e. 60666) component. This zip code extraction is done for each bus stop, rail line, and bus route to construct the zip code index. For reverse geocoding services, we use Yahoo’s Reverse Geocoding API [142].

6.6 Evaluation

In this section we discuss our training data collection procedure and the experimental results. We present the mode detection accuracy results when we ignore the transportation network, compared to detection accuracy results after we include transportation network information. Additionally, we use classification feature selection to rank our initial set of features. Given this ranking, we then select the highest rank features to build a final system.

6.6.1 Data Collection

For training the classifier, we collected traces on six different modes of transportation (walking, bus, car, stationary, aboveground train and bike). The data was collected by 6 individuals, 3 females and 3 males. The data was collected over a 3 week period. Additionally, three types of mobile devices were considered for data collection: (1) HP IPAQ PDA, (2) Android based Samsung Galaxy mobile phone (3), IPhone 3G. These devices are shown in Figure 75. Our application platform is the mobile web. Transportation mode ground truth was labeled on the GPS sensor reports by each individual using the user interface (UI) of the mobile web application (see Figure 76(a)). In Figure 76(b), we present a table, depicting the duration of mode specific labeled training data, collected from the six experiment participants.
6.6.2 Training data preprocessing

The accuracy of GPS varies. For example, GPS tends to underperform if it does not have a clear view of the sky (e.g. in urban canyons). For this reason, we perform a noise filtering step before training the classifier. Invalid GPS points are suppressed based on the GPS accuracy and the change in speed. GPS sensor reports with high inaccuracy readings and unrealistic changes in speed are pruned. This is a manual step before classifier training. GPS noise filtering before classifier training is not a new concept. The authors of [125] perform a preprocessing step before training their classifier.

GPS sensor reports are submitted by the mobile user from the mobile device every 15 seconds. A window size of 30 seconds was chosen as the period of classification. Therefore, for every two GPS sensor reports received, we constructed the classification feature set. We observed that submitting GPS reports is very power consuming. Thus, submitting GPS reports every second as done in [125] will exhaust battery power.
Figure 76: Data collection user interface. (a) User interface for data collection and ground truth labeling (b) Amount of training data collected

6.6.3 Window Size

In the proposed work, transportation mode detection accuracy was not sensitive to window size. However, larger window sizes result in longer transportation mode detection time.

6.6.4 Classifier selection

To determine the most accurate classifier for the proposed transportation mode detection algorithm, we compared precision and recall accuracy of five distinct classification models. The five models are: (1) Naive Bayes (NB), (2) Bayesian Network (BN), (3) Decision Trees (DT), (4) Random Forest (RF), (5) Multilayer Perceptron (ML). Readers are referred to [143, 144] for discussions on various classification models. To evaluate the different classification models on transportation mode detection, the WEKA machine learning tool set [143] was used.
The results indicate that Random Forest (RF) is the best model, with an average precision accuracy of 93.70% and recall of 93.80%. Thus, Random Forest classification system is chosen as the final classification model that is deployed to the public.

6.6.5 **Mode detection accuracy**

In this section, we analyze the performance and effectiveness of the transportation mode classifiers. We evaluate the classification schemes using two metrics: (1) **precision accuracy** and (2) **recall accuracy**.

\[
\text{Precision Accuracy (M)} = \frac{\text{number of correctly classified instances of mode M}}{\text{number of instances classified as mode M}}
\]

\[
\text{Recall Accuracy (M)} = \frac{\text{number of correctly classified instances of mode M}}{\text{number of instances of mode M}}
\]

Three sets of results were obtained and are presented in subsections 6.5.1, 6.5.2, and 6.5.3, respectively. Each set contains the precision accuracy and recall accuracy for the five classification schemes. For each set, we used 10-fold cross validation. In 10-fold cross validation, the original sample was randomly divided into 10 subsamples. Of the 10 subsamples, a single subsample was retained as the validation data for testing the classification model, the remaining 9 subsamples were used for training data. The cross-validation process was then repeated 10 times, with each of the 10 subsamples used exactly once as the validation data. The 10 results were then averaged to produce a single estimation.

6.6.5.1 **Classification without Transportation Network data**

Figure 77 shows the first set of results which are the precision accuracy and recall accuracy when transportation network related features are not considered. The only features considered are average speed, average acceleration, average heading change, and average GPS position accuracy. Thus, real time bus locations, rail line trajectory and bus stop locations are removed.
From Figure 77 it can be observed that Random Forest classification model is the most accurate model since it has higher average precision and recall accuracy compared to the other four classification models.

In general, when transportation network related features are not considered, the accuracy is below 76% for the five classification models (BN, NB, DT, RF, and ML). Additionally, we observe that motorized transportation and bikes show the lowest precision accuracy. For example, consider the case of Random Forest. The precision accuracy results for car, bus, and train are 58.1%, 56.5%, and 69.8%, respectively. The precision accuracy for bike is 71.4%. On the other hand, for non-motorized modes, such as walk and stationary, the precision accuracy is 100% and 96.8%. For all five models (BN, NB, DT, RF, and ML), the precision accuracy is best for walk and stationary. This implies that the transportation network data is not very helpful for detecting stationary and walk mode. On the other hand, features such as speed, heading, acceleration, and GPS accuracy are not sufficient for distinguishing between motorized modes because of feature similarities.

### 6.6.5.2 Classification with Transportation Network data

Figure 78 shows the second set of results which are precision and recall accuracy when all the classification features discussed in §6.5 are used. The main difference between Figure 77 and Figure 78 is that, in Figure 77, transportation network related classification features are not considered, while in Figure 78 transportation network related features are considered.

In Figure 78, all classifiers (BN, NB, DT, RF, ML) tested, with the exception of the Neural Network based Multilayer Perception (ML), achieve over 90% average precision and recall accuracy. On the other hand, when transportation network features are suppressed, the average precision and recall accuracy is below 76% (see Figure 77). This suggests that the transportation network related features are effective for transportation mode detection.
In the study, the most effective classification model is again Random Forest (RF), with an average precision accuracy of 93.7% and recall of 93.8%. This work is the first to distinguish between motorized transportation modes with such high accuracy [123, 124, 126, 127].

<table>
<thead>
<tr>
<th>Mode</th>
<th>Precision Accuracy</th>
<th>Recall Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NB</td>
<td>BN</td>
</tr>
<tr>
<td>train</td>
<td>70.0</td>
<td>50.0</td>
</tr>
<tr>
<td>bus</td>
<td>47.0</td>
<td>43.9</td>
</tr>
<tr>
<td>stationary</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>walk</td>
<td>94.7</td>
<td>93.8</td>
</tr>
<tr>
<td>car</td>
<td>42.3</td>
<td>90.0</td>
</tr>
<tr>
<td>bike</td>
<td>70.2</td>
<td>71.0</td>
</tr>
<tr>
<td>average</td>
<td>71.8</td>
<td>74.9</td>
</tr>
</tbody>
</table>

Figure 77: Transportation network features not considered.
From Figures 77 and 78, it can be seen that the precision and recall accuracy of motorized transportation modes and bikes increases more than the non-motorized modes of walk and stationary. For example, in the case of \textbf{RF} classification model, when transportation network features are used (Figure 78), the precision accuracies of car, train, bus, and bike are 87.5\%, 98.4\%, 88.3\% and 88.9\% respectively. On the other hand, when transportation network related features are not used (i.e. Figure 77), in the case of \textbf{RF}, the precision accuracies for car, train, bus, and bike are 58.1\%, 69.8\%, 56.5\% and 71.4\% respectively.

We conclude that our novel transportation network features are most effective for motorized transportation mode detection, and also effective for bike mode detection. This makes sense, since bikes and motorized modes may have similar speed and acceleration in traffic and therefore are difficult to distinguish using traditional motion pattern features. However, features of the transportation network, such as bus locations, help the distinguishing between buses and bikes.

Distinguishing among motorized transportation modes is useful in practice. For example, companies such as Google collect data from travelers’ mobile phones in order to estimate the traffic speed of a road segment. For this purpose the speed estimation system should only use the speed reports submitted by mobile devices on cars or buses but not those on trains. Distinguishing the train mode from the other motorized modes enables the speed estimation system to filter out speed reports submitted from trains.

The proposed approach is 17\% more accurate than \cite{124, 126} which uses GPS only and distinguishes between two motorized modes (bus and car). Also, the proposed approach is 9\% more accurate than \cite{123, 127} which uses GPS/GIS and distinguishes between bus and car. Using the newly proposed classification features, we show that we can detect transportation mode with high accuracy. These classification features are robust and most effective for detecting motorized transportation and bikes.
6.6.5.3 Transportation Mode Classification Feature Selection

Feature selection is a data-mining concept [145], which chooses the subset of input features by eliminating classification features that are less predictive. Using two commonly used feature selection algorithms, we ranked the eight classification features to identify the most relevant features for detecting transportation mode in the proposed work. The feature selection algorithms used are: (1) Chi Squared and (2) Information gain [145]. The ranking of the initial eight classification features are shown in Figure 79. Removing irrelevant classification features reduces the computational cost for training and transportation mode detection.

From Figure 79 we can see that the set of five top ranked classification features is the same for Chi Square and Information Gain. Thus, from Figure 79, we selected the five top ranked classification features, namely average speed, average acceleration, average rail line closeness, average bus closeness, and candidate bus closeness. We used these features to build a final classification model. The precision and recall accuracy of this final classification model is shown in Figure 80.

According to Figure 80, when the top five features are selected and the other three classification features are pruned, the precision accuracy hardly changes (see Figure 78 as well). Observe that Random Forest (RF) classification model is still the dominating classification model in Figure 80, with a precision and recall accuracy of 92.8% and 92.9% respectively. For RF, only a 0.9 % reduction in recall and precision accuracy is noticed when the five top ranked features are considered, as opposed to considering all eight classification features. This indicates that the top five classification features are enough to detect transportation mode in the proposed work.

In some cases, there is an increase in precision accuracy when only five features are considered. For example, consider the case of Random Forest precision accuracy for bus or bike transportation mode. For another example, the precision accuracy for DT when we consider the top five classification features is greater for cars and bikes, than if we consider all initial eight classification features.
In general, Figure 80 shows that even though we pruned three classification features, the accuracy is unaffected. This suggests that the three pruned features are redundant for detecting the transportation mode in the proposed work.

<table>
<thead>
<tr>
<th>Rank</th>
<th>Chi Squared</th>
<th>Information gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>average speed</td>
<td>average speed</td>
</tr>
<tr>
<td>2</td>
<td>average rail line closeness</td>
<td>average rail line closeness</td>
</tr>
<tr>
<td>3</td>
<td>average bus closeness</td>
<td>average acceleration</td>
</tr>
<tr>
<td>4</td>
<td>average acceleration</td>
<td>average bus closeness</td>
</tr>
<tr>
<td>5</td>
<td>candidate bus closeness</td>
<td>candidate bus closeness</td>
</tr>
<tr>
<td>6</td>
<td>average heading change</td>
<td>average heading change</td>
</tr>
<tr>
<td>7</td>
<td>average bus stop closeness</td>
<td>average bus stop closeness</td>
</tr>
<tr>
<td>8</td>
<td>average accuracy</td>
<td>average accuracy</td>
</tr>
</tbody>
</table>

Figure 79: Classification feature ranking and selection

<table>
<thead>
<tr>
<th></th>
<th>Precision accuracy</th>
<th>Recall accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NB</td>
<td>BN</td>
</tr>
<tr>
<td>train</td>
<td>96.5</td>
<td>92.2</td>
</tr>
<tr>
<td>bus</td>
<td>81.3</td>
<td>85.5</td>
</tr>
<tr>
<td>stationary</td>
<td>100</td>
<td>97.3</td>
</tr>
<tr>
<td>walk</td>
<td>94.6</td>
<td>94.7</td>
</tr>
<tr>
<td>car</td>
<td>79.2</td>
<td>82.2</td>
</tr>
<tr>
<td>bike</td>
<td>91.7</td>
<td>93.9</td>
</tr>
<tr>
<td>average</td>
<td>90.6</td>
<td>91.0</td>
</tr>
</tbody>
</table>

Figure 80: Only five high order classification features used.
The most effective features are average speed, average acceleration, average rail line closeness, average bus closeness, and candidate bus closeness.

Now we discuss Figure 79 from the perspective of transportation network data availability. Depending on its availability, the transportation network data can be categorized into three levels. The most widely available data is network topology data such as rail line routes. Figure 79 shows that this data is also most useful among transportation network features for mode detection. This is a good property of our approach. It means that our approach can be deployed to many regions in the world and is likely to achieve good performance there. The less widely available data is bus stop locations. Figure 79 shows that this data is least useful among the top ranked features. This means that our approach would not lose too much performance in the regions where bus stop information is unavailable. The least available data is real-time bus locations, which is a very predictive feature (i.e., average bus closeness) according to Figure 79. Thus our approach will not be able to utilize this predictive feature in many regions of the world. This is a limitation of our approach.

6.6.6 Performance and scalability

The speed at which we create the transportation mode classification features for training and inference is important. Recall, from the user’s GPS sensor reports, we need to compute the closest Euclidian distances to real time locations of buses, rail line trajectories, and bus stops.

There are over 11,500 bus stops in the city of Chicago. For larger cities such as New York, the number of bus stops may be even greater. Doing a linear comparison with all the bus stops, buses, and rail lines is time consuming. Doing a linear comparison took us over 2 minutes on a HP Laptop with a 4GB RAM and 2.54GHz Intel Core 2 Duo processor. This is impractical and ineffective in the real world, since in two minutes, users may transfer from one transportation mode to another, or become frustrated with the system. The proposed zip-code based indexing and pruning approach reduces our feature creation time from over 2 minutes to below 10 seconds. This can be further improved by using more sophisticated techniques, such as indexing by R-trees. However, performance was not a focus of this work.
6.6.7 **Extended real world evaluation**

The final classification model (Random Forest), using the top five ranked features, was deployed to the public via the mobile web. As explained earlier, we focused on the centralized server model. In this model, mobile users submit their GPS sensor reports via the web to our central server for classification. The central server then responds to the user with the corresponding transportation mode. Below, in Figure 81, we show the final deployed transportation mode classification system under operation in an IPhone 3G.

When the detected transportation mode is “bus” in Figure 81, we provide further information by giving the bus’s identification. The bus identification is a finer granularity of transportation mode detection than bus route; we can also detect the bus route on demand. This work is the first to infer such detailed transportation mode detection.

We also evaluated the deployed system to learn how the system performs in the real world under everyday usage. For evaluation purpose, new individuals that were not considered for the initial training data collection were given IPhone 3Gs with access to the classification system. We considered new individuals for this experiment, because we wanted to learn how the system would perform for new users that are not covered by the training data. These new users mounted the IPhone in any desired position (i.e. waist, arm, pocket, or bag), then tally the percentage of time the mode detection is correct. For example, if the mode was detected 8 out of 10 times correctly, the accuracy is 80%. The results of the real world evaluation for a mobile user are presented in Figure 82.
Figure 81: Deployed classification system

Figure 82 shows that, when deployed in the real world, under everyday usage, we achieved an average detection accuracy of 93.42% for the proposed mode detection system. The results indicate that the proposed approach is effective under everyday usage, and new training data collection is not necessary for new users. Also, the identified transportation network related features are very robust to traffic condition changes.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Duration (min)</th>
<th>Accuracy %</th>
</tr>
</thead>
<tbody>
<tr>
<td>train</td>
<td>35</td>
<td>93</td>
</tr>
<tr>
<td>bus</td>
<td>30</td>
<td>95</td>
</tr>
<tr>
<td>car</td>
<td>30</td>
<td>89</td>
</tr>
<tr>
<td>walk</td>
<td>30</td>
<td>92</td>
</tr>
<tr>
<td>bike</td>
<td>30</td>
<td>93</td>
</tr>
<tr>
<td>stationary</td>
<td>34</td>
<td>98.5</td>
</tr>
</tbody>
</table>

Figure 82. Evaluation of deployed system
6.6.8 **Bus mode detection discussion**

We presented three new classification features that can detect if a traveler is travelling via bus. The three features are: (1) average bus closeness (ABC), (2) candidate bus closeness (CBC), and (3) bus stop closeness rate (BSCR). From the feature selection in Figure 81, we observe that BSCR was overshadowed by ABC and CBC.

ABC captures the Euclidian distance to the closest bus for each snapshot. This Euclidian distance is summed over all the snapshots in a time window. Then, the average Euclidian distance is represented as ABC. ABC does not capture the traveler’s relationship with all the buses, only the closest bus.

CBC requires the knowledge of the Euclidian distances to all the buses. Then, the single closest bus over a time window is chosen as the candidate bus. Thus, CBC does not capture bus transfers. For example, if a traveler alights from a bus, and boards another bus, CBC may not identify the correct bus.

According to Figure 79, ABC is a more effective classification feature than CBC for transportation mode detection. In order to quantify the contribution of CBC to bus mode detection accuracy as a classification feature, we present bus mode accuracy results using the Random Forest Model, when the CBC feature is suppressed. The four high order features (average speed, average acceleration, average rail closeness, and average bus closeness) of Figure 79 are used in Figure 83. From Figure 83, we observe that the precision accuracy of buses decreases to 85.1% (Figure 83) from 89.7% (Figure 80), when only the top four features are used. This indicates that even though CBC may be more time consuming to compute, and use more memory than ABC, it is a worthwhile feature for buses. On the other hand, if speed and memory is critical; CBC can be suppressed, and bus mode detection will remain over 85% accurate.

<table>
<thead>
<tr>
<th></th>
<th>Precision Accuracy %</th>
<th>Recall Accuracy %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus</td>
<td>85.1</td>
<td>91.3</td>
</tr>
</tbody>
</table>
6.7 **Summary**

In this chapter we proposed a new robust approach to detecting transportation modes. In the proposed work, we considered and used transportation network data consisting of real time location of buses, rail lines, and bus stops spatial data. The real time location of buses is available in many cities such as Chicago, New York, Toronto, London, Washington DC, and San Francisco.

Using the transportation network data, we showed that it is possible to address the weakness of previously proposed solutions [123-127]; that is, to distinguish between motorized modes such as trains, buses, and cars with high accuracy. Furthermore, if we detect that a traveler is traveling by bus, we can further identify on which particular bus the person is traveling.

Among the five classification models considered, Random Forest model is the most dominating classification model with over 93% precision and recall accuracy. When transportation network classification features are not considered, the precision accuracy decreased to below 76%. This reduction of accuracy, upon omission of transportation network related features, is more notable for motorized transportation modes and bikes. This implies that transportation network data is effective for detecting motorized transportation, and bikes.

We also realized that, in order to achieve high precision and recall accuracy, only a subset of our initial set of classification features is necessary. In addition to traditional features on average speed and average acceleration, we identified for the first time the features on average bus closeness, average rail line closeness, and average candidate bus closeness. Using only this subset of features, and suppressing the other classification features that are not necessary, the precision accuracy was still over 92.5%.

Finally, users that have not participated in the initial training data collection evaluated the deployed system in the “real world” under everyday usage. The real world evaluation of the deployed system
resulted in a precision accuracy of 93.42%. This indicates that additional training data collection is not necessary for new users; and the system is robust under every day usage.
7 CONCLUSION AND FUTURE WORK

In this dissertation we proposed the MARKET algorithm for querying mobile P2P databases. MARKET includes a novel strategy for a mobile peer to prioritize the reports based on their relevance. The relevance of a report depends on its size, demand (how many peers are querying it), and supply (how many peers already have it). Queries are disseminated to enable the estimation of demand. A machine learning algorithm, called MALENA, is used to enable the estimation of the supply.

We compared MARKET with RANDI, LRU, and LFU by simulations using real-world mobility traces. The comparison was in terms of a metric that incorporates throughput and response time: the total number of reports matching the internal query that are delivered to an average consumer within a certain response-time bound. The results show that MARKET outperforms RANDI and LFU in all the test cases. MARKET outperforms LRU when the database size is small and the transmission size is big, whereas LRU outperforms MARKET when the database size is big and the transmission size is small. Furthermore, 75% of the reports which are accessible immediately in an ideal situation are also accessible within 10 minutes in an environment where the MARKET caching scheme is employed.

We instantiated the general-purpose MARKET algorithm in the context of discovering competitive physical resources. We determined that MARKET results in reduced discovery time compared to the case where no information is used. Sometimes the discovery time is cut by 70%. We studied the impact of various parameters on the value of resource information (i.e. the amount of discovery time saved by using the information). The results show that in MARKET mode, the value of resource information increases as the contention on resources increases. In addition, in MARKET mode the resource information is more valuable when the broker density and the transmission range are large. However, resource information is less useful as more consumers use it. We also determined that the motion speed has little impact on the value of resource information.

We developed and compared strategies for querying blob data in mobile P2P networks, where cellular communication is also available. In a systematic and exhaustive manner, we defined 13 possible query
processing strategies. We proved analytically that 4 of the 13 strategies are strictly better than the others. We compared the four non-dominated strategies by simulations in a vehicular environment. The simulations revealed that the best strategy is the one that separates metadata dissemination from its blob report, combines push of metadata and pull by queries, and uses the cellular infrastructure to communicate blobs.

We instantiated the MARKET algorithm in the context of continuous KNN query processing and compared it with DIKNN. The comparison shows that when the network is sparse (peer density = 1), DIKNN completely fails, with no correct answers returned at all, whereas MARKET manages to provide up to 30% accuracy. When the peer density is 2, MARKET outperforms DIKNN by an order of magnitude and provides up to 60% accuracy. When the peer density is higher than 5, DIKNN outperforms MARKET.

We studied the pattern of report propagation. The pattern shows that, by very simple local decisions made at each vehicle, the MARKET algorithm automatically limits the global distribution of a resource to a bounded spatial area. The spatial and temporal boundaries automatically adapt depending on the number of resources in the system, the traffic density and speed, and other parameters that dictate the amount of storage, processing power, and bandwidth that should be allocated to each resource. For example, if the number of resources is small, each resource will stay in the system longer, and spread farther.

For the future work, we plan to implement a client-server application that enables vehicles to upload and retrieve videos of real-time traffic conditions. In this application, a vehicle captures video clips of the traffic condition around it and uploads them to a central server. The vehicle may also query video clips from the central server, e.g., “Video clips of the road segment one mile ahead of me which are captured less than 1 minute ago”. This application is similar to the YouTube system. In fact, we will explore the feasibility of building it based on the YouTube service. There are several issues that need to be addressed. First, what is the best user interface for the user to input the query while driving? Second, can the bandwidth of the cellular infrastructure support the massive deployment of the application where millions of vehicles upload to and download from the central server concurrently? In the case that the bandwidth is
a problem, then how to reach the best tradeoff between the bandwidth consumption and the quality of service? We will deal with these issues in the future work.

**Other uncertainty models.** In this dissertation we assume that the uncertainty region of each object is a circle with a fixed radius. Other uncertainty models are possible. First of all, the radius may change per time, e.g., determined by the product of the maximum speed of the object and the time since the last location update (see [123]). For objects that move along straight line paths, the uncertainty region is a line-segment (see [118]). It is also possible that both the speed and the direction of the object change within a certain range. In this case, the uncertainty region is a fan area (see [126]). For these uncertainty models, many principles used in this dissertation, such as Proposition 5.3 for determining the \(k\)NN set, is still applicable. However, the shapes of max-curves and min-curves are different for different uncertainty models.

Another uncertainty model allows that there is location uncertainty associated with the query object as well. This happens, for example, when the query processing is performed at a central database, where the location is imprecise for all objects including \(O_q\). In this case, Proposition 5.3 does not hold anymore. Specifically, \(O_i\)'s min-curve being below the \(k\)-th level at a time \(t\) is only a necessary, but not a sufficient condition, for \(O_i\) to be a \(P_k\)NN at \(t\). For details see Appendix D. The implication of this observation is that when there is uncertainty associated with the query object the \(P_k\)NN set cannot be determined just based on the maximum and minimum possible distances. Thus, the answer-pairs probably cannot be computed merely in the Time-SquareDistance space; the relationship among the objects in the motion space needs to be examined as well. We notice that, however, existing studies [86, 94, 103, 104] apply Proposition 5.3 even though there is uncertainty associated with the query object in their models.

**Other query semantics.** In this dissertation we define a query to ask for possible \(k\)NNs. Another useful query semantics would be to ask for the objects that are *definitely* the \(k\)NNs regardless of the location configuration. When \(k=1\), a “definitely” query can be easily processed as follows. First we compute the \(P_1\)NN set (i.e., the set of objects that are possible 1NNs) using an algorithm introduced in this dissertation. If there is only one object in the \(P_1\)NN set, then that object is the answer to the
“definitely” query; otherwise, the answer to the “definitely” query is empty. However, this approach does
not straightforwardly extend to $k>1$. If there are exactly $k$ objects in the $PkNN$ set, it is still true that these
$k$ objects are the answer to the “definitely” query. However, if there are more than $k$ objects in the $PkNN$
set, we cannot say that the answer to the “definitely” query is empty because $k$ of these objects may be
definitely the $k$NNs.

**Indexing in the Time-SquareDistance space.** In existing studies, indexes are built in the motion
space (see [94, 104]). Would it be more efficient if we build indexes in the Time-SquareDistance? That is,
we index the distance curves using a spatial indexing structure such as a quadtree. A quick observation is
that the distance curves are invariable in time unless there are updates. On the other hand, the locations of
moving objects change continuously in the motion space even if there are no updates. Thus intuitively an
index structure in the Time-SquareDistances space would be more stable than one in the motion space.
CITED LITERATURE


http://crawdad.cs.dartmouth.edu/data.php


[64] Resource Description Framework. [http://www.w3.org/RDF/](http://www.w3.org/RDF/)

[65] SPARQL Query Language for RDF. [http://www.w3.org/TR/rdf-sparql-query/](http://www.w3.org/TR/rdf-sparql-query/)


[75] [http://wireless.agilent.com/rfcomms/refdocs/wcdma/e6703t_user_documentation.php](http://wireless.agilent.com/rfcomms/refdocs/wcdma/e6703t_user_documentation.php)


References


[120] Mountain Bike. [http://www.mtbroutes.co.uk/northyorkmoors/default.aspx](http://www.mtbroutes.co.uk/northyorkmoors/default.aspx)

[121] SportsDo. [http://sportsdo.net/Activity/ActivityBlog.aspx](http://sportsdo.net/Activity/ActivityBlog.aspx)


J. Hightower, S. Consolvo, A. LaMarca, I. Smith, J. Hughes. Learning and Recognizing the Places We Go. ACM Conference on Ubiquitous Computing, 2005.

Chicago Transit Authority http://www.transitchicago.com/about/facts.aspx

Chicago Transit Authority Bus Tracker http://www.ctabustracker.com/


Google Maps http://googlemaps.com/


APPENDICES

Appendix A: Proof of Proposition 5.2

**PROPOSITION 5.2:** Assume that the query object and the moving objects all move piecewise linearly, where each object can have at most \( m \) linear pieces. \( \beta_k(n) \geq 2m(n-k)+1 \).

**PROOF:** For the motion in Figure 84(a) the curves arrangement is as in Figure 84(b), and the details are as in Proposition 5.1. □

![Figure 84](image_url)

(a) Configuration in the motion space

![Figure 84](image_url)

(b) Arrangement in the Time-SquareDistance space

**Figure 84.** The auxiliary figure for the proof of Proposition 5.2.
Appendix B: Proof of Lemma 5.1

**Lemma 5.1.** Let $A$ be a connected sequence of $m_A$ parabola-pieces and $B$ be a connected sequences of $m_B$ $x$-monotone parabola-pieces, in the Time-SquareDistance space. There are at most $2(m_a + m_b - 1)$ intersections between $A$ and $B$.

**Proof.** Denote by $S_1$ the set of pieces in sequence $A$ that are intersected by the first piece in sequence $B$, by $S_2$ the set of pieces in $A$ that are intersected by the second piece in $B$, and so on. Denote by $|S|$ the size of a set $S$. Since any pair of pieces intersect at most twice, there are at most $2(|S_1| + |S_2| + \ldots + |S_{m_B}|)$ intersections between $A$ and $B$. On the other hand, due to the monotonicity of time, there is at most one common piece between $S_i$ and $S_{i+1}$. Thus, the total number of distinct pieces in $A$ that are intersected by $B$ is at least

$$|S_1| - 1 + |S_2| - 1 + \ldots + |S_{m_B}| = (\sum_{i=1}^{m_B} |S_i|) - (m_B - 1)$$

Since the number of distinct pieces in $A$ is $m_A$, the following inequality holds:

$$(\sum_{i=1}^{m_B} |S_i|) - (m_B - 1) \leq m_A$$

Thus, $\sum_{i=1}^{m_B} |S_i| \leq m_A + m_B - 1$. Thus there are at most $2(m_A + m_B - 1)$ intersections between $A$ and $B$. \(\square\)

Appendix C: Proof of Theorem 5.5

**Theorem 5.5.** Assume that the query object and the expected locations of each data object move piecewise linearly, where each object has at most $m$ pieces. Then for any constant $k$, $\widetilde{\mathcal{C}}_k(n) = \Omega(n^2 \cdot m)$.

**Proof:** We prove the theorem by constructing a feasible case in which the number of answer-pairs is quadratic in $n$ and linear in $m$. The construction proceeds as follows. Let object $O_q$ and the expected locations of $O_1$, $O_2$, ..., and $O_k$ be static in the motion space such that $O_i$ has the $i$-th maximum possible
distance to $O_q$ (see Figure 85(a)). Denote by $R_{i}^{\text{max}}$ the circle the center of which is the location of $O_q$ and the radius of which is the maximum possible distance between $O_i$ and $O_q$. Let the route of the expected location of object $O_{k+1}$ have $m$ linear pieces such that for each piece, the trace of $O_{k+1}$’s farthest-to-$O_q$-point intersects $R_{i}^{\text{max}}$ twice but does not intersect $R_{k-1}^{\text{max}}$, as shown in Figure 85(a). Let $O_{k+2}$ have the same route as $O_{k+1}$ and move behind $O_{k+1}$ such that its farthest-to-$O_q$-point enters $R_{k}^{\text{max}}$ after that of $O_{k+1}$ leaves $R_{k}^{\text{max}}$. Construct the same for $O_{k+3}, \ldots, O_{n/2}$. Figure 85(b) shows the max-curves of the first $k+1$ objects.

Denote by $R_{i}^{\text{min}}$ the circle the center of which is the location of $O_q$ and the radius of which is the minimum possible distance between $O_i$ and $O_q$. Let $O_{n/2+1}$ be static such that $R_{i}^{\text{min}}$ intersects the trace of $O_{k+1}$’s farthest-to-$O_q$-point for $2m$ times (see Figure 85(c)). Do the same construction for $O_{n/2+2}, \ldots, O_n$. Figure 85(d) shows how the min curves intersect the max k-level. It is not difficult to see that there are

\[2 \cdot \frac{n}{2} \cdot m \cdot \left(\frac{n}{2} - k\right)\]
critical intersections.\[\square\]
(b) Max-curves of the first $k+1$ objects.

(c) Configuration of $O_{n/2+1}$ in the motion space.

(d) Intersections between the min-curve of $O_{n/2+1}$ and the max $k$-level.

Figure 85. The Concrete Example for the Proof of Theorem 5.5
Appendix D: Applicability of Proposition 5.3 when there is location uncertainty associated with the query object

When there is location uncertainty associated with the query object $O_q$, Proposition 5.3 does not hold anymore. Specifically, $O_i$’s min-curve being below the $k$-th level at a time $t$ is only a necessary, but not a sufficient condition, for $O_i$ to be a $P_k$NN at $t$. This is because the distances of different objects to the query object are not independent to each other; they depend on which location within the query object’s uncertainty region is used to compute the distance. This concept is illustrated by the following simple example. Figure 86 shows the uncertainty regions of the query object $O_q$ and the only two moving objects $O_a$ and $O_b$, at a time $t$. The radius is $r$ for all the uncertainty regions. According to the figure, the maximum possible distance between $C_a(t)$ and $C_q(t)$ is $4r$. The minimum possible distance between $C_b(t)$ and $C_q(t)$ is smaller than $4r$. Thus $O_b$’s min-curve is below $O_a$’s max-curve at time $t$. If we apply Proposition 5.3, we would conclude that $O_b$ is a P1NN at time $t$, but this is obviously wrong. The reason is that the query object $O_q$ can be at only one location at a time. Specifically, in order for $O_q$ to get the maximum distance to $O_a$, $O_q$ has to take the location at $P$ in Figure 86. On the other hand, in order for $O_q$ to get the minimum distance to $O_b$, $O_q$ has to take the location at $P'$. But $O_q$ can only take one location at time $t$.

Figure 86. Proposition 5.3 does not hold when there is location uncertainty associated with the query object.
CURRICULUM VITAE

Education

• Ph.D. in Computer Science, IGERT Computational Transportation Science Program, University of Illinois at Chicago, 2012
• Ph.D. in Computer Science, University of Electronic Science and Technology of China, 1997
• Master in Computer Science, University of Electronic Science and Technology of China, 1994
• Bachelor in Computer Science, Beijing University of Aeronautics and Astronautics, 1991

Academic Appointments

• 2012–present: Argonne National Laboratory, Transportation Research and Analysis Computing Center, Postdoctoral Research Associate
• 2004–2012: University of Illinois at Chicago, Department of Computer Science, Research Scientist
• 1997–2000, 2002–2004: University of Illinois at Chicago, Department of Computer Science, Postdoctoral Research Associate
• 1997: University of Electronic Science and Technology of China, Lecturer

Industrial Appointments

• 2000–2002: Armillaire Technologies, Member of Technical Staff, Team Leader, Chicago, IL

Teaching Experience

• Database Systems, full-semester course, Fall 2012, University of Illinois at Chicago, (co-teaching with Professor Ouri Wolfson)
• Object-oriented Programming, one-week full-time course, June 1996, IBM, China
• Computer Network Programming, full-semester course, Spring 1996, Spring 1997, University of
Electronic Science and Technology of China

- **Software Development Environment: UNIX**, full-semester course, Fall 1995, University of Electronic Science and Technology of China

- **C Programming**, one-week full-time course, May 1992, Science and Technology Information Research Institute of Sichuan Province, China

**Honors & Awards**


- **Achievement in Professional Excellence**, IBM Corp., 1996, for teaching “Object-oriented Programming”.

- **Best Software Award (first prize)**, Science and Technology Committee of Sichuan Province, China, 1994.

- **Best Software Award (second prize)**, Science and Technology Committee of Sichuan Province, China, 1996.

- **Teaching Excellence Award**, University of Electronic Science and Technology of China, 1996, for teaching “Computer Network Programming”.


- **Distinguished Ph.D. Student Fellowship Award**, University of Electronic Science and Technology of China, 1996.


- **Distinguished Undergraduate Student Fellowship Award**, Beijing University of Aeronautics and Astronautics, 1987-1990.

**Publications**

**Theses**


**Articles in Refereed Journals**


9. B. Xu, P. Xiong, J. Liu, Objected-oriented Technology in OS/2 User Interface, *Computer*
Applications, Vol. 16, No. 6, 1996, pp. 3-5.*


**Articles in Conference Proceedings (Refereed)**


* Publications written in Chinese.


41. Ouri Wolfson, Bo Xu, Huabei Yin and Hu Cao. Searching Local Information in Mobile Databases (Poster Paper). In *Proceedings of the 22nd International Conference on Data Engineering (ICDE)*, Atlanta, Georgia, USA, April, 2006.


**Articles in Conference Proceedings (Non-refereed)**


Sections in Books and Bulletins


Patent


Research Grants

- National Science Foundation Grant 0847680, O. Wolfson (PI), B. Xu (Co-PI), SGER: Feasibility of Decentralized Search in Mobile P2P Databases, $100,000, 9/08-8/09.
- National Science Foundation Grant OII-0611017, O. Wolfson (PI), B. Xu (Research Institute PI), STTR Phase I; Feasibility of Mobile Peer-to-Peer Search on Hand-held Devices, $99,982, 7/06-6/07 (12% success rate).
- NASA STTR Award NNA06AA25C, O. Wolfson (PI), B. Xu (Research Institute PI), MOBI-DIC: MOBIle Dissemination of loCal information in Peer-to-Peer Wireless Networks, $99,940, 1/06-1/07 (15% success rate).
- National Science Foundation Grant ITR-0086144, Ouri Wolfson (PI), Bo Xu (Co-PI), ITR: Real-time
Capture, Management and Reconstruction of Spatio-Temporal Events, $673,837, 9/00-9/04. This ITR project is in collaboration with UCLA, University of Maryland, and UC Irvine, and $673K is UIC’s share.

**Oral Presentations**

- “Query Processing in Mobile Peer-to-peer Databases”, Université de Valenciennes et du Hainaut Cambrésis, France, December 15, 2011.
- “Query Processing in Mobile Peer-to-peer Databases”, INRIA, Lille, November 28, 2011.
- “In-network Query Processing in Mobile P2P Databases”, *17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems* (ACM GIS), Seattle, WA, November 4-
“Query Processing in Mobile Peer-to-peer Databases”, UIC CTS-IGERT Seminar, June 18, 2009.


“The Role of Auto-ID Technologies in Mobile Databases for E-Commerce”, International Workshop on RFID Data Management, Cancun, Mexico, April 7, 2008.


“DOMINO: Databases for Moving Objects tracking”, demo presentation, 1999 ACM SIGMOD
Software Systems Developed

- MOBI-DIK – A mobile peer-to-peer database for local search. MOBI-DIK development was funded by NASA and NSF.
- DOMINO – Databases fOr MovINg Objects tracking. DOMINO adds temporal, uncertainty management, and simulation capabilities to a database management system, for tracking the location of vehicles and aircraft. The system was invited for demonstration at the ACM-SIGMOD’99, NGITS’99, and ICDE’01 conferences.
- LAN Management System – This SNMP-based network management system provides topology discovery, management information viewing, remote station controlling, and event monitoring.
- LAN Messenger System – This system provides a network messaging function for sending messages to and receiving messages from other users on a TCP/IP network.
- PCB CAD/CAM – Eliminating photo-plotting and etching which are required in normal PCB process and are usually handled by a subcontractor, this system allows the production of PCB prototype in hours instead of days, and thus helps shorten time-to-market.

Professional Activities

- Program Vice Chair, First International Workshop on Knowledge and Data Engineering in Web-based Learning, December 8-10, 2010, Shanghai, China
- Chair, Second International Workshop on Information Management for Mobile Applications, August 31, 2012, Istanbul, Turkey
- Organizer, NII Shonan Meeting: Social Issues in Computational Transportation Science, December, 2012, Shonan, Japan

Member of the program committee for the following conferences:

- Second International Workshop on Mobile P2P Network, June 28 – July 2, 2010, Gaen, France
- 2009 Mobile P2P Workshop, June 21-24, 2009, Leipzig, Germany
• Second International Workshop on Computational Transportation Science, November 3, 2009, Seattle, USA

• First International Workshop on Computational Transportation Science, July 21, 2008, Trinity College Dublin, Ireland

Referee for the following Journals:

• IEEE Transactions on Intelligent Transportation Systems
• IEEE Transactions on Vehicular Technology
• IEEE Transaction on Knowledge and Data Engineering
• IEEE Transaction on Parallel and Distributed Systems
• IEEE Transactions on Systems, Man, and Cybernetics
• GeoInformatica
• Journal of High Speed Networking
• Wireless Networks
• Information Sciences
• International Journal of Autonomous and Adaptive Communications Systems
• Transportation Research Part C

Referee for the following conferences:

• ACMGIS’11, 19th International Conference on Advances in Geographic Information Systems, November 1-4 2011, Chicago, Illinois.

• SSTD’11, 12th International Symposium on Spatial and Temporal Databases, August 24-26, 2011, Minneapolis, MN, USA.

• MDM’11, 12th International Conference on Mobile Data Management, Lulea, Sweden, June 6-9, 2011.

• ICDE’11, 27th IEEE International Conference on Data Engineering, April 11-16, 2011, Hannover, Germany.
• VLDB’10, 36th International Conference on Very Large Data Bases, September 13-17, 2010, Singapore.

• ICDE’10, 26th IEEE International Conference on Data Engineering, March 1-6, 2010, Long Beach, California, USA.

• MDM’09, 10th IEEE International Conference on Mobile Data Management, May 18-20, Taipei, Taiwan.

• ICDE’09, 25th IEEE International Conference on Data Engineering, March 29 – April 2, 2009, Shanghai, China.

• ACM-GIS’08, 16th ACM International Symposium on Advances in Geographic Information Systems, November 5-7, 2008, Irvine, CA, USA.

• MDM’08, 9th IEEE International Conference on Mobile Data Management, April 27-30, 2008, Beijing, China.

• SIGMOD’08, ACM International Conference on Management of Data, June 10-12, 2008, Vancouver, Canada.


• MDM’07, 8th IEEE International Conference on Mobile Data Management, May 7-11, 2007, Mannheim, Germany.

• ACM-GIS’06, 14th ACM International Symposium on Advances in Geographic Information Systems, November 10-11, 2006, Arlington, VA, USA.

• VLDB’05, 31st International Conference on Very Large Databases, August 30 – September 2, 2005, Trondheim, Norway.

• SSTD’05, 9th International Symposium on Spatial and Temporal Databases, August 22-24, 2005, Angra dos Reis, Brazil.

• SIGMOD’05, ACM International Conference on Management of Data, June 13-16, 2005, Baltimore,
Maryland, USA.

- ICDE’05, 21th IEEE International Conference on Data Engineering, April 5-8, Tokyo, Japan.
- ICDE’04, 20th IEEE International Conference on Data Engineering, March 30 – April 2, 2004, Boston, MA, USA.
- DASFAA’04, 9th International Conference on Database Systems for Advanced Applications, March 17-19, Jeju Island, Korea.
- MobiDE’03, Third ACM International Workshop on Data Engineering for Wireless and Mobile Access, September 19, 2003, San Diego, CA, USA.
- VLDB’03, 29th International Conference on Very Large Databases, September 9-12, 2003, Berlin, Germany.
- CIKM’02, 11th ACM International Conference on Knowledge and Data Engineering, November 4-9, 2002, McLean, VA, USA.
- MDDS’02, 5th International Workshop on Mobility in Databases & Distributed Systems, September 2-6, 2002, Aix-en-Provence, France.
- VLDB’02, 28th International Conference on Very Large Databases, August 20-23, 2002, Hong Kong, China.
• ACM-GIS’00, 8th ACM International Symposium on Advances in Geographic Information Systems, November 10-11, 2000, Washington D.C., USA.

• VLDB’00, 26th International Conference on Very Large Databases, September 10-14, 2000, Cairo, Egypt.


• TeleGeo’00, Second International Workshop on Telegeoprocessing, May 10-12, Nice-Sophia-Antipolis, France.

• WNMC’00, Workshop on Wireless Networks and Mobile Computing in conjunction with The 20th International Conference on Distributed Computing Systems, April 2000, Taipei, Taiwan.

• ACM-GIS’99, 7th ACM International Symposium on Advances in Geographic Information Systems, November 2-6, 1999, Kansas City, USA.


• MobiDE’99, ACM International Workshop on Data Engineering for Wireless and Mobile Access, August 20, 1999, Seattle, WA, USA.

• MDSD’99, Second International Workshop on Mobility in Databases & Distributed Systems, August, 1999, Florence, Italy

• SSD’99, 6th International Symposium on Spatial Databases, July 20-23, 1999, Hong Kong, China.


• TeleGeo’99, First International Workshop on Telegeoprocessing, May 6-7, Lyon, France.


Professional Certificates
- CISCO Certified Network Associate
- IBM Certified Specialist AIX V4.1 System Administration
- IBM Certified VisualAge for Smalltalk Associate Developer
- Novell Education Certificate: UNIX OS Fundamentals for NetWare Users, UnixWare Installation and Configuration, UnixWare System Administration, UnixWare Advanced System Administration, NetWare NFS, NetWare TCP/IP