Distributed Spatial Skyline Query Processing in Wireless Sensor Networks *

SunHee Yoon and Cyrus Shahabi

Department of Computer Science, University of Southern California
sunheeyo@usc.edu, shahabi@usc.edu

Abstract. Spatial skyline queries can be used in wireless sensor networks for *collaborative positioning* of multiple objects. However, designing a distributed spatial skyline algorithm in resource constrained wireless environments introduces several research challenges: how to distribute the computation of distances to multiple events in order to compute the skylines efficiently, accurately, quickly, progressively, and concurrently while dealing with the network and event dynamics. We address these challenges by designing, implementing, and extensively evaluating the Distributed Spatial Skyline (DSS) algorithm. DSS is the first *distributed* algorithm to compute spatial skylines. In a network of 554 nodes, DSS reduces the communication overhead by up to 91% over a centralized algorithm.

1 Introduction

Given a set of data points P and a set of query points Q, *spatial skylines* are those members of P that are not spatially dominated by any other point in P with respect to Q [19, 24]. A point $p1$ spatially dominates a point $p2$ with respect to Q, if and only if $p1$ is closer to at least one query point as compared to $p2$ and has the same distance as $p2$ to the rest of the query points. The spatial skyline query can be utilized to *collaboratively position* multiple targets in wireless networks. For example, suppose from a set of police cars (data points P) we want to identify a candidate subset to be dispatched to multiple incident points (query points Q). This candidate subset includes those cars that are not dominated by any other police car with respect to all the incidents, and hence they are the spatial skylines. There are two ways to find the candidate police cars, one is to have a centralized server with all the updated locations of cars and incidents and then perform the spatial skyline query. An alternative is that the police cars identify the candidate sets themselves by local communication.

* This research has been funded in part by NSF grants IIS-0238560 (PECASE), IIS-0742811, CNS-0831505 (CyberTrust), and the NSF Center for Embedded Networked Sensing (CCR-0120778). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

In this paper, we propose a distributed algorithm to collaboratively find the spatial skylines in wireless sensor networks, and show that the distributed approach is more efficient than the centralized approach. Because the centralized approach assumes a complete set of data in space and time, if we use TAG-like data collection tree [13], it requires high transmission overhead for resource constrained wireless networks. Moreover, for the time-critical applications, the centralized approach cannot meet the promptness requirement because it does not provide any partial results, *i.e.* progressive results, early. We address these challenges by proposing a Distributed Spatial Skyline (DSS) algorithm to collaboratively find the spatial skylines in wireless sensor networks. DSS parallelizes the search for skylines by partitioning the search space based on the geometric properties of the nodes and the topology. This approach enables an efficient, scalable, progressive, fast, and concurrent search while providing correct results. In a network of 554 nodes, the DSS algorithm outperforms the centralized algorithm by up to 91% while providing 100% accurate results progressively. Our DSS algorithm is the first *distributed* algorithm to compute *spatial skylines*.

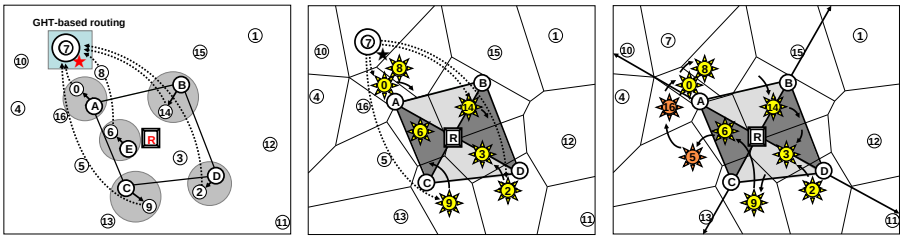
This paper is organized as follows. Section 2 presents research related to DSS, and Section 3 describes the DSS algorithm. Section 4 presents the results from our experimental evaluations. Section 5 concludes our work and presents our future work.

2 Related Work

Over the past several years, skyline computation has become a popular research topic in the database research community in areas such as multi-criteria decision making and data mining [1, 4, 6–8, 11, 12, 14, 15, 18–21].

Most of existing skyline algorithms, however, are centralized algorithms that work on static [4, 8, 11, 12, 14, 15] or streaming datasets [4, 12, 14, 15, 18, 19]. Although, several distributed skyline algorithms [2, 5–7, 21, 23, 25] have been proposed, they are studied in the context of either vertically [2] or horizontally [21, 25] partitioned datasets over connected networks such as peer to peer networks [21] or MANETs [5, 7]. Wu *et al.* proposed the Distributed SkyLine query (DSL) algorithm [23] that uses Content Addressable Network (CAN). DSL pipelines the skyline query execution over a large number of machines by leveraging the content-based data partitioning. Cui *et al.* proposed a Parallel Distributed Skyline query processing (PaDSkyline) algorithm [6] for distributed systems by partitioning and filtering. Zhu *et al.* proposed a Feedback-based Distributed Skyline (FDS) algorithm [25] in which the server iteratively filters the dominated data distributed over systems. A few distributed skyline algorithms have been proposed for wireless networks. Huang *et al.* proposed a distributed skyline algorithm for MANETs [7]. Chen *et al.* proposed a threshold-based skyline monitoring algorithm for sensor networks [5]. However, there is no known *distributed* algorithm to compute *spatial skyline*.

Abstract regions [22] provides a set of spatial operators by utilizing region-based collective communication, and simplifies application development in sensor



(a) Phase 1 Computing query convex hull and R.

(b) Phase 2 Search internal skylines.

(c) Phase 3 Search external skylines.

Fig. 1. Three phases of DSS algorithm. {A,B,C,D,E} are query points and quadrilateral $ABDC$ is the query convex hull. Star is a synchronization point, R is the rendezvous point, and sunflowers are spatial skylines = {0,8,14,2,3,9,6,5,16}.

networks. Although several abstract regions are implemented, an operator for spatial skyline is still missing. Implementing efficient distributed spatial skyline using operators proposed in abstract regions is challenging because spatial skylines cannot be defined within a certain number of hops or distance from a node. Computing correct spatial skylines may require the location information beyond the neighborhood relationship on which abstract regions are built.

3 DSS Algorithm

The DSS algorithm partitions the network into two sets of nodes: nodes inside or outside of *query convex hull*, $CH(Q)$, which is the convex hull of query points, Q. DSS secondarily partitions those two sets of nodes by triangulation; Let *triangle*, T, be the triangle region divided by two consecutive points in $CH(Q)$ and *rendezvous point*, R, which is the centroid of $CH(Q)$. Let *extended triangle*, ET, be the union of T and the outside area of $CH(Q)$ confined by the extended line connecting R and two consecutive points in $CH(Q)$ (but closed area by the network boundary).

The DSS algorithm assumes that 1) each node has computed a list of its Voronoi neighbors using a well-known distributed Voronoi computation algorithm [3], 2) there are more than three query points without a degenerated case where three points are positioned on a line, and 3) all the nodes are connected by a single spanning tree so that they are mutually routable. DSS algorithm operates in these three phases:

Phase 1: In phase 1, DSS computes $CH(Q)$ and R for partitioning. The node closest to each query point reports the <coordinate of the query point> using GHT-based routing [17] to R (star in Fig 1(a)), where all the query points across the network are gathered. GHT home, the closest node to R, (node 7 in Fig 1(a)) receives those coordinates, and computes $CH(Q)$ and R. DSS then transmits the $<CH(Q), R>$ to each *triangle home*, t, which is the node closest to each point in $CH(Q)$.

Phase 2: In phase 2, DSS searches the internal skylines in each T which are definite skylines located within or near $CH(Q)$. Each t (nodes 0, 14, 2, and

9 in Fig 1(b)) naturally becomes a spatial skyline by the definition [19]. t subsequently searches for other undiscovered internal skylines among all Voronoi neighbors by applying two rules [19][1] with $CH(Q)$ replaced by triangle[2]; DSS determines a node to be an internal skyline either 1) if a node is within a triangle, or 2) if the Voronoi cell of a node intersects the edge not adjacent to R in the triangle (*e.g.* the edge AB in the triangle ABR in Fig 1(b)). For this internal skyline search, DSS utilizes *Voronoi-based geographic flooding*, which is a technique to multicast a packet to all the Voronoi neighbors and enables efficient communications with all the nodes in a given geographical region. In this service, a node first performs 1-hop radio transmissions reaching all the Voronoi neighbors within the radio transmission range. Those Voronoi neighbors that are farther than 1-hop transmission range are reached by multihop paths using GPSR [9]. When a node determines that it is a skyline, it reports itself as a skyline to t using GPSR. When t receives all the internal skylines from its triangle, it sends this list to R. In Fig. 1(b), the triangle home 0 reports skylines {0,8}, 14 reports {14}, 2 reports {2,3}, and 9 reports {9,6} to R. Now R has all the internal spatial skylines {0,8,14,2,3,9,6} which is sent to the user to provide progressive results before DSS identifies the remaining skylines in phase 3.

Phase 3: In phase 3, DSS searches external skylines, the skylines that are located outside of $CH(Q)$ (Fig. 1(c)). Each t traverses the skylines of corresponding ET found during phase 2 by sending $<CH(Q)$, triangle points, internal skylines$>$ to them. While this message traverses each skyline, DSS discovers a new skyline if any of its Voronoi neighbors is not spatially dominated by the skylines in the packet. DSS appends the new skyline to the list, and sends it to the next skyline node in the list. This traversal terminates when no new skylines are discovered [19]. Then, DSS sends the list of skylines to the clockwise t. When the clockwise t receives all the skylines, it sends this list to R. In Fig. 1(c), each clockwise t 0, 14, 2, and 9 report the aggregated skylines {9,6,5,16}, {0,8}, {14}, and {2,3} to R, and R aggregates them into the final spatial skyline list {0,8,14,2,3,9,6,5,16}.

4 Evaluation

In this section, we describe the evaluation metrics, experimental setup, and results.

4.1 Evaluation Metrics and Experimental Setup

We compare the DSS algorithm to the centralized spatial skyline algorithm running over TAG [13]-like data collection tree using the following metrics: 1) *Number of transmissions (ntx)* which estimates the energy cost of the algorithm in wireless sensor networks, 2) *Accuracy of results* defined as the percentage of

[1] These two rules are proved in the centralized spatial skyline algorithm [19].

[2] This substitution does not impact the correctness of the algorithm because the union of all the triangles is equal to $CH(Q)$ and DSS searches all the triangles.

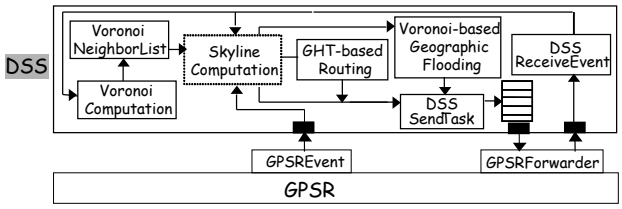


Fig. 2. Software architecture of DSS.

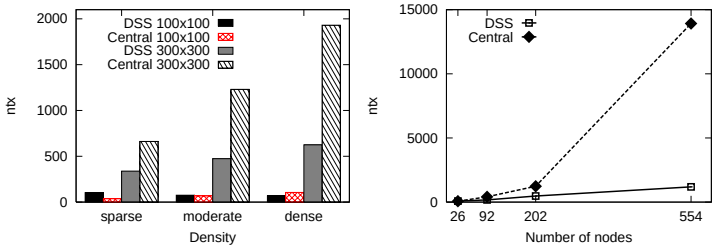
actual skylines (gathered by DSS) over the correct skylines, 3) *Progressiveness* defined as the delay until the first and the first 50% of the skylines are computed, and 4) *Delay to compute skylines* defined as the time of receiving the complete skylines.

We implemented DSS in TinyOS 1.1.15 [10]. We performed simulations by changing parameters such as network density, size, number of query points, MBR(Q), and synchronization point locations and performed a total of 25,200 runs across these parameter combinations. All the results are averaged over 10 different topologies with the same parameters. Fig 2 presents the software architecture of DSS over GPSR [9]. We use GPSR for both GPSR routing and GHT-based routing.

In each of our experiment, the query points are positioned at *i.i.d.* random location with uniform distribution while satisfying the given MBR(Q). We placed the centroid of the MBR(Q) at the centroid of the network. All the nodes are positioned at *i.i.d.* random location with uniform distribution across the network. All the nodes in the network are connected; they form a single spanning tree verified by Kruskal's algorithm. We used qhull library [16] for Voronoi neighbor computation. We made sure that there are no degenerated cases with query points less than three (cannot form a query convex hull) or any three nodes on a straight line. We use the disc radio model with a communication radius of 50 meter. We do not simulate packet losses due to the interference or buffer overrun by GPSR [9], but we simulate the packet drops due to the routing failure. All protocols used in DSS are robust to packet failures except the Voronoi-based geographic flooding in phase 2.

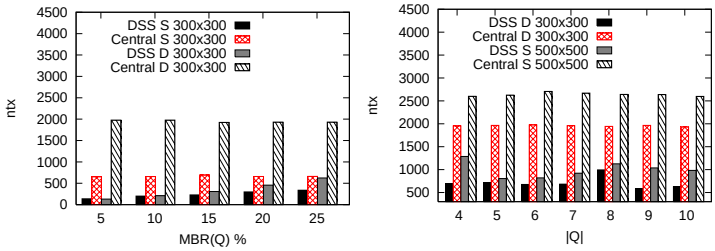
4.2 Efficiency and Scalability

DSS is more efficient than the centralized algorithm in large networks. Fig. 3(a) shows the number of transmissions incurred by DSS and the centralized algorithm as we increase the network size and density. We observed that DSS outperforms the centralized algorithm as the network becomes denser or larger. In a large and dense network with 292 nodes in 300×300 meter, DSS incurs 68% fewer transmission compared to the centralized algorithm. DSS has 91% fewer transmissions than the centralized algorithm in a large network with 554 nodes in 500 × 500 network (Fig. 3(b)). DSS is more efficient than the centralized algorithm because only a skyline node transmits a packet when it reports the results while all the nodes transmit packets at least once in the centralized algorithm.



(a) By network size and density: 100×100 (16, 26, 36 nodes), 300×300 (108, 202, 292 nodes).

(b) By the number of nodes: 26, 92, 202, 554 nodes in moderate 100×100, 200×200, 300×300, 500×500 network respectively.



(c) By MBR(Q): Sparse and dense 300×300 (108, 292 nodes).

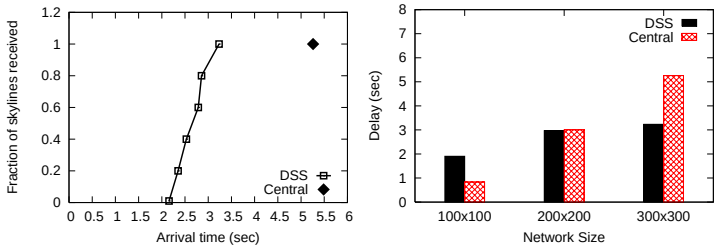
(d) By |Q|: 292 nodes in dense 300×300 and sparse 500×500.

Fig. 3. Message transmission overhead of DSS with |Q| = 10 and MBR(Q) = 25% unless mentioned specifically. D denotes dense and S denotes sparse.

DSS is more scalable than the centralized algorithm. Fig. 3(b) shows the number of transmissions as we increase the size of network. The communication overhead of DSS scales linearly with the size of the network while the overhead of centralized algorithm scales exponentially.

The number of transmissions by DSS does not depend on the number of query points but depends on the *area of the query convex hull*. Fig. 3(c) shows that the number of transmissions by DSS increases with larger MBR(Q) while the number of transmissions by the centralized algorithm is almost constant over different MBR(Q) areas. Fig. 3(d) shows that the number of transmissions by DSS does not show any dependence on the number of query points. This result validates the property of spatial skyline such that the query points inside of query convex hull do not have any impact on the spatial skylines. The spatial skylines are determined by the geometric relation between nodes and the query convex hull.

With the same number of nodes, DSS is more efficient in denser networks than in sparser networks. Fig. 3(d) shows that with 292 nodes, DSS in denser network (300×300) uses 36% less transmissions than in sparser network (500×500) with |Q| = 10. DSS's efficiency in denser network is due to relatively short routing paths. Particularly, the Voronoi-based geographic flooding used in phase 2 enables DSS to traverse the internal query convex hull of denser and smaller networks more efficiently, thereby resulting in the shorter routing path.



(a) Progressiveness (300×300) (b) Delay to compute skyline

Fig. 4. Progressiveness and delay of the DSS algorithm, with fixed |Q| = 10, MBR(Q) = 25%, and moderate density

4.3 Accuracy and Completeness

The DSS algorithm provides 100% accurate and complete skylines across different parameters: network sizes and densities. We validated this result by comparing the result returned by DSS against the result returned by the centralized algorithm which is proven to provide 100% accurate skylines.

4.4 Progressiveness and Delay to Compute Skylines

Unlike the centralized algorithm, DSS provides partial results early before the full results are available. Fig. 4(a) shows the fraction of skylines received by DSS along the arrival time. DSS returns the first skyline to R at 2.16 second, 50% skylines by 2.65 second, and the final skylines at 3.23 second. However, the centralized algorithm does not return any partial results until the final skylines are computed at 5.26 second. DSS provides linear progressiveness from the start to the end; DSS shows similar progressiveness until near 80% of skylines are returned which are the internal skylines, and 20% of skylines, which are the external skylines, arrive soon thereafter.

Fig. 4(b) shows that the delay to compute skylines with the DSS algorithm. Although the delay by DSS is worse in small networks compared to that of the centralized algorithm, it is comparable to or better than that of the centralized algorithm as the network scales. As the network size grows, this difference becomes larger. Because the triangulation in both internal and external query convex hull enables parallel execution, DSS can speed up the skyline search.

5 Conclusions and Future Work

We designed DSS, the first distributed algorithm to compute spatial skyline in wireless sensor networks. Spatial skyline is a useful primitive to select multiple positions for tracking or other collaborative positioning applications. We demonstrated that the DSS algorithm is more efficient, more scalable, and faster than the centralized algorithm while providing 100% accurate skylines progressively. We plan to extend our work to compute distributed spatial skylines with mobile query points and sensor nodes.

References

1. S. Antony, P. Wu, D. Agrawal, and A. E. Abbadi. MultiObjective OLAP: Efficient Skyline Computation over Ad-Hoc Aggregations. In *ICDE*, Apr. 2008.
2. W.-T. Balke, U. Gntzer, and J. X. Zheng. Efficient Distributed Skylining for Web Information Systems. In *EDBT*, Mar. 2004.
3. B. A. Bash and P. J. Desnoyers. Exact distributed Voronoi cell computation in sensor networks. In *IPSN*, Apr. 2007.
4. C.-Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang. Finding k-Dominant Skylines in High Dimensional Space. In *SIGMOD*, June 2006.
5. H. Chen, S. Zhou, and J. Guan. Towards Energy-Efficient Skyline Monitoring in Wireless Sensor Networks. In *EWSN*, June 2007.
6. B. Cui, H. Lu, Q. Xu, L. Chen, Y. Dai, and Y. Zhou. Parallel Distributed Processing of Constrained Skyline Queries by Filtering. In *ICDE*, Apr. 2008.
7. Z. Huang, C. S. Jensen, H. Lu, and B. C. Ooi. Skyline Queries Against Mobile Lightweight Devices in MANETs. In *ICDE*, Apr. 2006.
8. M. Khalefa, M. Mokbel, and J. Levandoski. Skyline Query Processing for Incomplete Data. In *ICDE*, Apr. 2008.
9. Y.-J. Kim, R. Govindan, B. Karp, and S. Shenker. Geographic Routing Made Practical. In *NSDI*, May 2005.
10. P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications. In *SenSys*, Nov. 2003.
11. X. Lian and L. Chen. Monochromatic and Bichromatic Reverse Skyline Search over Uncertain Databases. In *SIGMOD*, June 2008.
12. X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting Stars: The k Most Representative Skyline Operator. In *ICDE*, Apr. 2007.
13. S. Madden, M. J. Franklin, J. Hellerstein, and W. Hong. TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks. In *OSDI*, Dec. 2002.
14. M. Morse, J. M. Patel, and H. Jagadish. Efficient Skyline Computation over Low-Cardinality Domains. In *VLDB*, Sept. 2007.
15. J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic Skylines on Uncertain Data. In *VLDB*, Sept. 2007.
16. qhull@qhull.org. Qhull. In *http://www.qhull.org/*.
17. S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin, , and F. Yu. Data-Centric Storage in Sensornets with GHT, A Geographic Hash Table. In *MONET, Vol.8 No.4*, 2003.
18. N. Sarkas, G. Das, N. Koudas, and A. K. H. Tung. Categorical Skylines for Streaming Data. In *SIGMOD*, June 2008.
19. M. Sharifzadeh and C. Shahabi. The spatial skyline queries. In *VLDB*, Sept. 2006.
20. Y. Tao, X. Xiao, and J. Pei. SUBSKY: Efficient Computation of Skylines in Subspaces. In *ICDE*, Apr. 2006.
21. A. Vlachou, C. Doulkeridis, M. Vazirgiannis, and Y. Kotidis. SKYPEER: Efficient Subspace Skyline Computation over Distributed Data. In *ICDE*, Apr. 2007.
22. M. Welsh and G. Mainland. Programming Sensor Networks Using Abstract Regions. In *NSDI*, Mar. 2004.
23. P. Wu, C. Zhang, Y. Feng, B. Y. Zhao, D. Agrawal, and A. E. Abbadi. Parallelizing Skyline Queries for Scalable Distribution. In *EDBT*, Mar. 2006.
24. S. Yoon and C. Shahabi. Poster Abstract: A Distributed Algorithm to Compute Spatial Skyline in Wireless Sensor Networks. In *IPSN*, Apr. 2009.
25. L. Zhu, Y. Tao, and S. Zhou. Efficient Distributed Skyline Retrieval. In *TKDE*, 2008.