2009 Tenth International Conference on Mobile Data Management: Systems, Services and Middleware

Operator Placement for Snapshot Multi-Predicate Queries in Wireless Sensor Networks

Georgios Chatzimilioudis [#1], Huseyin Hakkoymaz [#2], Nikos Mamoulis [*3], Dimitrios Gunopulos [#4]

[#] *Computer Science Department, University of California, Riverside*
Riverside, California 92507, USA
[1]gchatzim@cs.ucr.edu [2]huseyin@cs.ucr.edu [4]dg@cs.ucr.edu

[*] *Computer Science Department, Hong Kong University*
Pokfulam Road, Hong Kong
[3]nikos@cs.hku.hk

***Abstract*— This work aims at minimize the cost of answering snapshot multi-predicate queries in high-communication-cost networks. High-communication-cost (HCC) networks is a family of networks where communicating data is very demanding in resources, for example in wireless sensor networks transmitting data drains the battery life of sensors involved. The important class of multi-predicate queries in horizontally or vertically distributed databases is addressed. We show that minimizing the communication cost for multi-predicate queries is NP-hard and we propose a dynamic programming algorithm to compute the optimal solution for small problem instances. We also propose a low complexity, approximate, heuristic algorithm for solving larger problem instances efficiently and running it on nodes with low computational power (e.g. sensors). Finally, we present a variant of the Fermat point problem where distances between points are minimal paths in a weighted graph, and propose a solution. An extensive experimental evaluation compares the proposed algorithms to the best known technique used to evaluate queries in wireless sensor networks and shows improvement of 10% up to 95%. The low complexity heuristic algorithm is also shown to be scalable and robust to different query characteristics and network size.**

I. Introduction

There is a need for intersecting lists of horizontally or vertically distributed data in various high communication cost (HCC) network applications. The intersection can take place at the sink or in-network, while the lists are routed towards the sink. Advances in the most commonly researched HCC networks, namely Wireless Sensor Networks, have made it possible to store large amounts of data on the sensors and reduce communication by transmitting data in batch [1][2].

Multi-Predicate Queries (MP-queries) are queries that consist of different predicates and are answered by the intersection of the individual answers for each predicate. In Figure 1 we can see a visual example of a sink issuing an MP-query whose predicates are answered by some source nodes. An instance of an MP-query: *"Find the timestamps for which the temperature in region A and region B was 100, and the humidity in region C was 90%"*. More complex instances of MP-Queries are Spatio-Temporal Pattern (STP) queries, proposed and analyzed in [3]: *"Find object IDs that crossed through region A at time T_1, came through area B at a later time T_2 and then stopped inside circle C some time during interval (T_3,T_4)"*.

We focus on snapshot instances of MP-Queries and the optimization of their in-network processing. A Dynamic Programming algorithm (*DPopt*) is proposed that does an exhaustive search for the possible routing trees and intersection operator placement and always finds the optimal solution. An optimal heuristic to branch and bound on the search space of *DPopt* is proposed. This heuristic is based on a variant of the Fermat point problem that has many practical application in various network problems. Due to its high complexity, *DPopt* is not useful for large problem instances and especially not for sinks with low computational power. For this reason we developed a low complexity suboptimal heuristic algorithm *2PH* which is shown to perform very close to the optimal solution. The basic intuition is that the transmission of big lists should be avoided and smaller lists should be used to intersect with them.

The goal of this paper is to minimize energy expenditure needed to perform in-network evaluation of snapshot multi-predicate queries in HCC networks. We take a holistic approach looking at all operators and optimize both their sequence and their placement. Our contributions:

- We address the important class of multi-predicate queries for high communication cost networks
- We analyze the complexity of identifying the optimal plan for executing a multi-predicate query and prove that the problem is NP-hard.
- We propose a dynamic programming algorithm *DPopt* to compute the optimal plan
- We present an important variant of the Fermat point problem, propose a solution and use it as a branch and bound method to reduce the search space and improve the efficiency of the dynamic programming algorithm.
- We propose a fast suboptimal heuristic algorithm with performance very close to the optimal.
- An extensive experimental evaluation compares our algorithms to the best known technique used to evaluate queries in wireless sensor networks, and shows that savings from 10% up to 95% are possible. Our simple, heuristic algorithm is also shown to be scalable and robust to different query characteristics and network sizes.

978-0-7695-3650-7/09 $25.00 © 2009 IEEE
DOI 10.1109/MDM.2009.13

Our algorithms are general in the sense that they can be used for any kind of query involving a set of intersections. Also, they can easily be used within the core of previously proposed query optimizers for WSN (e.g. tinyDB [4]). To our knowledge, this is the first work to propose, develop and analyze algorithms for optimizing in-network processing of multi-predicate queries in high communication-cost networks.

The paper is organized as follows. Section II describes some of the related research and section III presents the problem definition, the assumptions and the cost model used. Sections IV and V discuss the complexity of the problem and present an algorithm for the optimal solution. Section VI describes the suboptimal heuristic algorithm and its optimization, whereas section VII presents the variant of the Fermat point problem and our strategy of using it to improve the efficiency of both optimal and suboptimal algorithms. Next, a combination of the algorithms is used to create a hybrid for better results in section VIII. The algorithms we compare to and the experimental evaluation follow in section IX. Section X summarizes and concludes the paper.

II. Related Work

Motivated by boolean queries in text database systems, works [5][6][7][8] consider minimizing the comparisons needed to intersect a collection of sorted lists. Lower and upper bounds for intersecting two lists have been analyzed, but those bounds correspond to intersecting lists by exchanging individual elements. We stay at the granularity of exchanging whole lists amongst nodes, which is shown by our experiments to be more cost efficient.

A. Distributed Databases

Semi-joins are by far the most widely studied and used technique in the area of distributed query processing with focus on minimizing communication cost. Semi-joins are shown to be of benefit only for relations with very large tuples [9][10]. Chang in [11] proposes heuristic rules for considering the optimal sequence of joining distributed relations. However, his heuristics account only for uniform networks and do not account for single attribute relations. The work of [12] is also using joins to reduce transmitted data. Their divide and conquer heuristic algorithm is developed for uniform networks and requires queries that involve both 'data reducing' and 'data increasing' join operators on different attributes in order to work. In our case we deal only with intersections which is always 'data-reducing' and which is defined between the same common attribute over all relations. Using their algorithm in our problem would return a random query tree.

Optimization techniques for queries have also been studied by Hellerstein [13] where an algorithm is proposed for the optimal ordering of selections. Like theirs, our algorithms are also motivated by the Least-Cost Fault Detection problem [14], which puts selection predicates in inversely proportional order of their selectivity. The heuristic proposed in [13], however, can not be straightforwardly applied in our setup since the differential cost in our case depends on the path chosen so far for computing the intersection.

B. Wireless Sensor Networks

Meliou et al. [15] consider gathering data from a small subset of sensors, where they propose a new query routing scheme where the query dissemination and the query evaluation are combined in one step. A fixed-size packet with the query is routed over all nodes of interest gathering their readings, and returned to the sink. The assumption that the size of the data gathered from each source is known is only viable in very specific applications. Furthermore, their algorithm does not account for the reduction imposed by intersecting two sets of data from different sources, thus it is not optimally applicable in our scenario.

Coman et al. [16] propose heuristics to optimize the placement of only a single binary join operator. In general, they pick the closest node to the optimal Euclidean location of joining. As we will show in this paper this heuristic is very dependent on the network layout and can result in a very costly plan. Yu et al. [17] are also interested in minimizing communication cost for a single binary equi-join of tuples coming from two different neighborhoods. They use synopsis to cut down on the data sent from each local neighborhood to the actual join operator and they compare against a method identical to what Coman et al. proposed in [16]. In our paper we want to optimize queries with more than one operator, thus we need to take a holistic approach looking at all operators and optimize both their sequence and their placement. If we assume that each predicate is answered by a neighborhood and not just by one node then we can incorporate the synopsis method by [17] in our framework for every operator placement.

III. Assumptions, Notation and Definitions

Several assumptions on the high-communication-cost network are needed to better analyze the theoretical aspect of the problem and study the behavior of the proposed methods. We consider that the nodes in the network are stationary throughout the query injection and answer retrieval. Also, the network is symmetric, which means that if a node u is able to listen to node v then v can also listen to u.

The basestation or querying node (called also *sink*) has full knowledge of the network which is a valid assumption if we use Link State routing (for example [18]). This assumption can be dropped in the case the nodes are aware of their location and the query consists of spatial predicates, since we can use geographic routing [19]. The way each site collects the data it needs is orthogonal to this work. Also, techniques to store and index data [20][21] are complementary to this work.

We assume that we are able to estimate the selectivity of each intersection operator. The efficiency of the proposed query optimization techniques depend on the accuracy of this estimation. A zero cost solution is to use past query answers to estimate the selectivity of new intersections. Another solution is using end-biased samples [22] that provide high accuracy with small communication cost for correlated data. Techniques

like histograms [23], sketches [24] and Bloom filters [25] have also been proposed and applied to estimate the selectivities of operators in a distributed system. Nevertheless, dealing with large datasets the cost of estimating the intersection selectivity is very low compared to the cost of query processing.

We will stay at the granularity of exchanging lists and not elements. As experiments show our algorithms always outperform exchanging elements.

TABLE I
NOTATION

N	set of all nodes in the wireless sensor network
n	number of nodes in the network $n = \|N\|$
Q	multi-predicate query
P	set of the predicates $p_1, p_2, ..., p_m$ in query Q
m	number of predicates in query Q, $m = \|P\|$
q	querying node (sink)
S	set of source nodes, each holding a single predicate answer A_i. $S \subset N$
A_i	set of values held in source node s_i that is the answer to some predicate p_j
B_u	Bytes to be sent from node u
$Hops(u, v)$	Hops of shortest path from node u to node v
$D(u, v)$	Distance function used as a heuristic in our algorithms

Multi-predicate snapshot queries: Our techniques target applications that make use of multi-predicate snapshot queries. Snapshot queries are instant queries that require their answer at the specific time of the query expression. Multi-predicate queries consist of different predicates, that are simple queries like selections, range, skylines, etc. The answer to the multi-predicate query is computed by the intersection of the predicate answers.

Query evaluation cost: We follow the same energy model as Coman et al [16] which can be applied even for the case where each predicate is answered by a region and not just one source node. The cost of transmitting data from node i to node j is $c(i, j) = c * Hops(i, j) * B_i$. The term c is network specific and independent of the query Q and the nodes i,j.

We define $k(i, j)$ as the function that returns 1 whenever the edge (i, j) is used in the query evaluation:

$$k(i,j) = \begin{cases} 1 & \text{if communication edge } (i,j) \text{ is used} \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

The total communication cost C_Q of getting the answer of query Q will be:

$$C_Q = \sum_{i=1,j=1}^{m,m} c * k(i,j) * B_i \quad (2)$$

where nodes $i, j \in V$. This does not include the cost of disseminating the query. We are only concerned about the cost of returning the results to the sink rather than the cost of disseminating the query itself.

We are trading communication cost for computation cost. It has been well documented though that computational cost is orders of magnitude lower than communication cost. The energy consumption for transmitting 1 bit of data using the MICA mote (http://www.xbow.com/) is approximately equivalent to processing 1000 CPU instructions. Also, in many real world applications WSN are multi-tier, thus trading cheap computational tasks at some high tier node for a low energy query plan that will be executed on low tier nodes of high energy constraints is very preferable.

Problem Definition: Consider a wireless sensor network with a set N of n nodes $n_1, n_2, ..., n_n$. We have a multi-predicate query Q – consisting a set P of m predicates – injected at node $q \in N$. Each predicate is assigned to one node, so there is a subset S of N with m nodes that can answer the query. Each node $n_i \in S$ is called source *and holds a predicate answer A_i. Our goal is to minimize the cost-function (2) of computing $\bigcap_{i=1}^{m} A_i$ and sending the result to the sink q.*

IV. NP-HARDNESS

Consider the simple case where all predicate answers are identical, thus any intersection of two sets will result in the same set again. In this case all the edges in our virtual graph will have static weight since the load transmitted over it will always be the same.

The definition of the Steiner Tree problem in graphs:

Steiner Tree in Graphs
Instance: Graph $G = (V, E)$, a weight $w(e) \in Z_0^+$ for each $e \in E$, a subset $R \subseteq V$, and a positive integer bound B.
Question: Is there a subtree of G that includes all the vertices of R and such that the sum of the weights of the edges in the subtree is no more than B?

Thus, a special case of our problem is already reducible to the Steiner Tree problem, which is proved to be NP-hard by Karp [26]. Thus, our problem is NP-hard also.

V. OPTIMAL SOLUTION

The optimal solution can be constructively found by a Dynamic Programming algorithm that we will call *DPopt*. During *DPopt*, we need to compute a $(2^m - 1) * n$ matrix C. Every row of this matrix corresponds to a subset of S (S has m source nodes), and every column is a node in N (N has n nodes). Each cell $c_{i,j}$ of the matrix C contains the minimum communication cost to get the intersection of subset i at node j. For example, $c_{\{s1s2\},n6}$ corresponds to the minimum cost needed to compute the intersection of A_1 and A_2 and get the result at n_6. As another example, $c_{\{s1s2s3\},n4}$ corresponds to the minimum possible cost to get the intersection of A_1, A_2, and A_3 at node n_4. In this case, n_4 could receive $A_1 \cap A_2 = A_{12}$ from one node and A_3 from another and compute $A_{12} \cap A_3$, or n_4 could receive $A_1 \cap A_3 = A_{13}$ from one node and A_2 from another and compute $A_{13} \cap A_2$, or n_4 could receive $A_2 \cap A_3 = A_{23}$ from one node and A_1 from another and compute $A_{23} \cap A_1$, or could even receive the whole intersection $A_1 \cap A_2 \cap A_3 = A_{123}$ from a different node. The goal of *DPopt* is to find which of the above ways is the cheapest and store its cost at $c_{\{s1s2s3\},n4}$

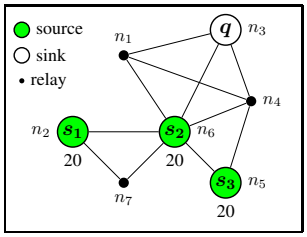

Fig. 1.

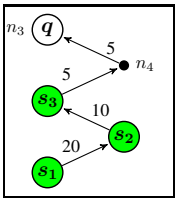

Fig. 2.

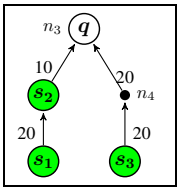

Fig. 3.

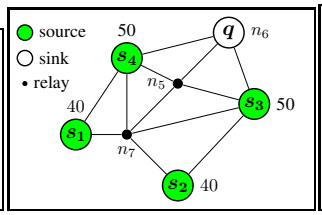

Fig. 4.

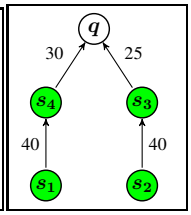

Fig. 5.

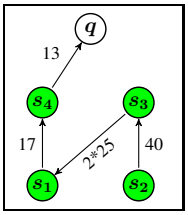

Fig. 6.

Fig. 1: An instance of a Sensor Network
Fig. 2: The optimal query evaluation tree given by the *DPopt* algorithm
Fig. 3: The query evaluation tree given by the *Naive*algorithm
Fig. 4: Another instance of a Sensor Network
Fig. 5: Query plan produced by Phase 2 for evaluation tree of Figure 7
Fig. 6: Query plan produced by by Phase 2 for evaluation tree of Figure 8

(and the corresponding sub-plan). How to create the Dynamic Programming matrix C for *DPopt* is described in detail later.

Assume that C has been completed. The last row of this matrix will store, for each node in N, the minimum cost of that node being the final recipient of the complete intersection. The cost in $c_{\{s1s2s3\},q}$ will correspond to the cost of the optimal plan for answering the MP-Query Q.

In matrix C each row corresponds to a subset $S_i \subset S$ and is constructed as follows. For each cell $c_{S_i,v}$ we consider each possible split of S_i into two sets S_{i1} and S_{i2}. We access the corresponding rows for S_{i1} and S_{i2}, and we measure the cost that these two nodes send the partial intersections to v. We also consider the whole set S_i being sent directly from a different node. Thus we need to consider all columns of the same row which might not yet be filled out. We compute the minimum of these costs and store it to $c_{S_i,v}$ (together with the corresponding sub-plan in a separate data structure). If this new cost affects other columns in the same row we update them. The computational cost of evaluating query Q according to our *DPopt* is mainly:

$$c_{S_i,v} = c_{S_{i1},u} + c_{S_{i2},w} + B_u Hops(u,v) + B_w Hops(w,v)$$

where S_{i1} and S_{i2} the non-overlapping splits of S_i and $u, v, w \in S$.

Table II shows the Dynamic Programming matrix C for our example in Figure 1 where the sources s_1,s_2,s_3 have cardinality 20 and every time we intersect we get $1/2$ of the smallest input elements. The first two rows are easy to complete: there are no possible splits of the single-element sets thus for each set we compute the cost of sending it to the node in each column. For instance, cell $(1,1)$ represents cost $c_{\{s1\},n1}$ of sending set $\{s_1\}$ to node n_1, 20 bytes traveling over 2 hops $n_2 \rightarrow n_6 \rightarrow n_1$ makes 40. In the same manner the other cells of rows 1-3 are computed. For rows 4-6 we can split the two-element sets only one way. For instance, set $\{s_1s_3\}$ can be only split into set $\{s_1\}$ and set $\{s_2\}$. $c_{\{s1s2\},n1}$ represents the optimal cost of sending the result of set $\{s_1s_2\}$ to node n_1. To find this optimal cost we need to test all possible ways of computing the result of $\{s_1s_2\}$. The optimal solution is sending both to node n_1 with costs $c_{\{s1\},n1} = 40$ and $c_{\{s2\},n1} = 20$ read from cells $(1,1)$ and $(2,1)$ respectively. This gives a total of 60 which is written into cell $(4,1)$. The rest of the cells in rows 4-6 are computed in the same way. For row 7, where we have the 3-element set $\{s_1s_2s_3\}$, we follow the same steps, only this time we have 3 possible splits $\{s_1s_2\}$-$\{s_3\}$, $\{s_1s_3\}$-$\{s_2\}$, and $\{s_2s_3\}$-$\{s_1\}$. For each possible split we need to test all scenarios of getting the result of each subset and send it to the appropriate node x, write the minimum cost into cell $(7,x)$ and store the plan used in a separate structure for reference. For $x = n_3$, which is the sink, we can get subset $\{s_1s_2\}$ with cost 30 as indicated by $c_{\{s1s2\},n3}$ and subset $\{s_3\}$ with cost 40 as indicated by $c_{\{s3\},n3}$ for a total cost of 70. An alternative is to get subset $\{s_1s_3\}$ with cost 50 and subset $\{s_2\}$ with cost 20 for a total cost of 70. Another possible split is getting subset $\{s_2s_3\}$ with cost 30 and subset $\{s_1\}$ with cost 40. We can also get the whole set $\{s_1s_2s_3\}$ from node n_2 with a total cost of 40, which is also the optimal solution that is written in $c_{\{s1s2s3\},n2}$. We can see that there are two optimal solutions: $s_3 \rightarrow s_2 \rightarrow s_1 \rightarrow s_2 \rightarrow q$ and the one shown on figure 2 both with cost 40.

TABLE II
DYNAMIC PROGRAMMING MATRIX C FOR *DPopt*

	n_1	n_2	n_3	n_4	n_5	n_6	n_7
$\{s_1\}$	40	0	40	40	40	20	20
$\{s_2\}$	20	20	20	20	20	0	20
$\{s_3\}$	40	40	40	20	0	20	40
$\{s_1s_2\}$	30	20	30	30	30	20	30
$\{s_1s_3\}$	50	40	50	50	40	40	50
$\{s_2s_3\}$	30	30	30	30	20	20	30
$\{s_1s_2s_3\}$	45	30	40	30	30	35	35

It is easy to prove that *DPopt* is an optimal algorithm for our problem definition, since it searches the solution space systematically and exhaustively. We can also show that the computational cost of *DPopt* is given by the following expression:

$$\sum_{i=1}^{m} \sum_{k=1}^{\frac{i}{2}} \left[n^3 \binom{m}{i} \binom{i}{k} \right] \quad (3)$$

where m is the number of predicates in the query and n is the number of nodes in the network. That is, for each i-length combination S_i of source nodes, one row of the matrix has to be constructed, and for each of the n cells in this row, for each of the splits of S_i into two sets S_{i1} and S_{i2}, n^2 pairs of cells must be examined from the corresponding previous rows of the matrix. It holds that

$$\sum_{i=1}^{m} \binom{m}{i} = 2^m - 1 \text{ and } \sum_{k=1}^{\frac{i}{2}} \binom{i}{k} < \sum_{k=1}^{i} \binom{i}{k}$$

we can simplify the complexity expression of our *DPopt* with the upper bound

$$\sum_{i=1}^{m}\sum_{k=1}^{i}\left[n^3\binom{m}{i}\binom{i}{k}\right]$$

$$= n^3\sum_{i=1}^{m}\left[\binom{m}{i}(2^i-1)\right] = n^3 * (3^m - 2^m - 1)$$

Thus, the time complexity of *DPopt* is $O(n^3 * 3^m)$. The space complexity is equivalent to the size of the matrix $O(n * 2^m)$.

VI. Heuristic Approximate Algorithm

DPopt is too slow and computationally demanding for large problems. In this section, we propose a suboptimal 2-phase heuristic algorithm (*2PH*) that operates much faster and in most of the cases finds a solution very close to the optimal. This algorithm is breaking up the problem into two phases: 1) find an *evaluation tree* that dictates the operator sequence, and 2) iteratively optimize the operator placement on this *evaluation tree*.

Distance Function: Our distance function is based on the cost function (2) given in section III and is the following:

$$D(u,v) = H_{uv} * B_u \tag{4}$$

, where H_{uv} is the hop-distance between nodes u and v, and B_u are the number of bits to be transmitted from u.

Phase 1: Simple bottom-up (agglomerative) hierarchical clustering is used. First, for each source-node a single-element cluster is initialized containing the source (note that the sink is not involved in the clustering process). Each cluster C has a load B_C, which is the estimated result size of the intersection of the lists of its children C_1 and C_2.

$$B_C = |\bigcap\{C_1, C_2\}| \tag{5}$$

Each cluster also has a representative node denoted as $C.repr$ with a representative load B_C. This node plays the role of the sink for the sources in the cluster. $C.repr$ is chosen over the sources composing cluster C only, and is the source that minimizes the cost

$$x = Hops(C_1.repr, s) * B_{C_1} + Hops(C_2.repr, s) * B_{C_2}$$

The result of *Phase 1* is the *evaluation tree* where each of the m leaves holds a single-element cluster containing a source and the root holds a cluster composed of all sources. For example the *evaluation* tree for the network in Figure 4 can be seen in Figure 7.

The distance of two clusters is determined by the distance of their representative nodes expressed by function (4). For resolving ties we propose some heuristic tie breaking rules:

1) Prefer the pair that is farthest from the sink. Data is likely to travel from the outer sources towards the sink.
2) Prefer the pair that will result in the smallest intersection. This way the next transmission is going to be cheaper.

Clusters are merged according to single linkage and the dimensionality of the problem space is low, thus the time complexity for Phase 1 is $O(m^2)$ [27], where m is the number of predicates in the query.

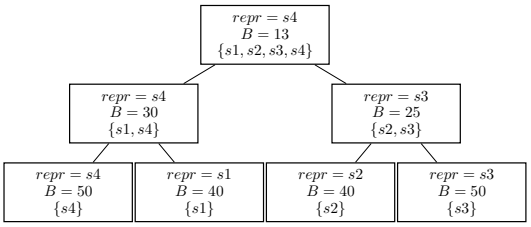

Fig. 7. *Evaluation* tree returned from Phase 1 without the optimization for network in Figure 4

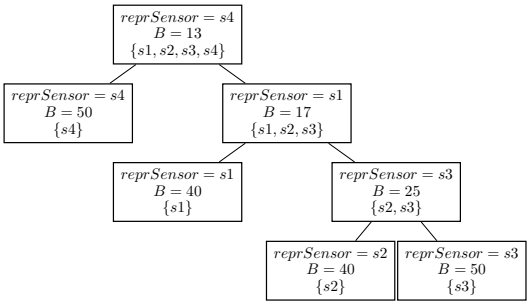

Fig. 8. Evaluation tree returned from Phase 1 with the optimization for network in Figure 4

Phase 2: The task here is to optimize operator placement given the *evaluation tree* of *Phase 1*. The *evaluation tree* is traversed top-down to optimize the operator placement for each internal tree-node. At each step we consider two source nodes s_1, s_2 and one destination node $dest$ and try to find the optimal operator placement node f that minimizes $Hops(s_1, f) * B_{s1} + Hops(s_2, f) * B_{s2} + Hops(f, dest) * B_f$. Starting from the root of the *evaluation tree* we decide where to place the operator for the current tree-node. The algorithm can be seen in Algorithm 1.

Algorithm 1 2PH_Phase2($dest$)

1: **for** each child C of $dest$ **do**
2: C_1 = left child of C
3: C_2 = right child of C
4: find node f that minimizes $Hops(C_1.repr, f) * B_{C1} + Hops(C_2.repr, f) * B_{C2} + Hops(f, dest) * B_f$
5: set node f as $operNode$ of C
6: call 2PH_Phase2(C)
7: **end for**
8: **return**

The algorithm scans the whole tree to determine $operNode$ for each internal node of the tree which takes $O(\log m)$ for balanced trees and $O(m)$ for unbalanced trees. To perform step 3 in Algorithm 1 we need to check all network nodes which is $O(n)$. The time complexity for Phase 2 is thus $O(n * m)$, where n is the number of nodes in the network and m is the number of predicates in the query Q. The total complexity of *2PH* is $O(n * m)$.

A. Optimization by Forcing Deep Trees

The algorithm described above has a major drawback: the *evaluation tree* it produces is very often bushy whereas the optimal solution is usually a left-deep tree connecting all the sources in a chain and fully exploiting the reducibility of the intersection operator. To take this into consideration we propose an optimization for *Phase 1* of the *2PH* algorithm. Before merging two clusters C_1 and C_2 we first check if it is more profitable to put one cluster C_1 as a child of the other cluster C_2 instead of having them as siblings (see Algorithm 2). Assume that $B_{C1} < B_{C2}$ where B_{C1} is the load of

Algorithm 2 2PH_Merge(C_1,C_2)

1: **if** $B_{C1} < B_{C2}$ **then**
2: $siblingCost = B_{C1} * Hops(C_1.repr, C_2.repr) + B_C * Hops(C_2.repr, q)$
3: **else**
4: $siblingCost = B_{C2} * Hops(C_2.repr, C_1.repr) + B_C * Hops(C_1.repr, q)$
5: **end if**
6: **if** $B_{C1} < B_{C2}$ **then**
7: $descendantCost$ = *Descendant_Cost*(C_1,C_2)
8: **else**
9: $descendantCost$ = *Descendant_Cost*(C_2,C_1)
10: **end if**
11: **if** $siblingCost > descendantCost$ **then**
12: C = make C_1 child of C_2
13: **else**
14: C = merge C_1 and C_2 as siblings
15: **end if**
16: **return**

cluster C_1 and B_{C2} is the load of cluster C_2. Then the cost $siblingCost$ of the resulting plan when two clusters are made siblings is

$$siblingCost = B_{C1} * Hops(C1.repr, C2.repr) + B_C * Hops(C2.repr, q)$$

where C is the cluster resulting from merging C_1 and C_2, B_C is the size of the intersected lists given by Equation 5, and q is the sink.

The cost of making cluster C_1 child of cluster C_2 is equal to:

$$descendantCost1 = B_{C1} * Hops(C_1.repr, s) + costDifference + B_C * Hops(C_2.repr, q) \quad (6)$$

where $costDifference$ is the difference in cost of creating cluster C_2 since now its members that lie on the path from s to $C_2.repr$ are going to be intersected with the list C_1, thus transmitting less data amongst them. This makes $costDiffernece$ always negative. The cost of equation 6 is computed by function *Descendant_Cost()* shown in algorithm 3. The node containing source s that minimizes the cost of transferring the data from $C_1.repr$ to C_2 is used as parent of $C_1.repr$ (line 1). Then, the evaluation tree of C_2 is updated so that the nodes on the path from s to $C_2.repr$ get merged with the new attached cluster C_1 and their load recalculated (lines 4-7). During this update the cost difference achieved from the updated loads is computed as $costDifference$ (line 7). For example, the *evaluation* tree for the network in Figure

Algorithm 3 Descendant_Cost(C_1,C_2)

1: s = source from C_2 that minimizes $B_{C1} * Hops(C_1.repr, s)$
2: $treenode$ = leaf node containing s
3: $costDifference$ = 0
4: **while** $treenode$ is not the root **do**
5: $oldB = B_{treenode.cluster}$
6: merge C_1 to $treenode.cluster$
7: $costDifference += (oldB - B_{treenode.cluster}) * Hops(treenode, treenode.parent)$
8: **end while**
9: **return** $B_{C1} * Hops(C_1.repr, s) + costDifference + B_C * Hops(C_2.repr, q)$

4 can be seen in Figure 7 and the corresponding query plan in Figure 6. This heuristic greatly improves performance as experiments show. We call this version *2PHdeep*.

The complexity of *Descendant_Cost()* depends on step 1 and the while loop of steps 4-6. The while loop is a bottom-up tree traversal from one leaf to the root which takes $O(\log m)$ time and Step 1 is $O(m)$. We can maintain a *kd-tree* for the source nodes and get s in $O(\log m)$ instead.

Lemma 1: The overall time complexity for the *2PHdeep* algorithm is $O(m^2 \log m)$.

VII. General Fermat Point in Graph

In both algorithms proposed so far we deal with the same subproblem: given two source nodes s_1, s_2 and a sink node q in a graph find the optimal node f to place the intersection operator in order to minimize the communication cost. So far in our algorithms we have been using exhaustive search to find node f. In this section we will propose some heuristics to cut down on the search space for f and accelerate all proposed algorithms.

The subproblem mentioned above is the building block of both our algorithms and we will call it *General Fermat Point in a Graph* and the optimal node will be called *Fermat node*. To the best of our knowledge the *General Fermat Point in Graphs* has not been identified prior to this work.

> *General Fermat Point in Graphs (GFPG)*
> Instance: Graph $G = (V, E)$, three nodes n_1, n_2, n_3 with weights w_1, w_2, w_3 respectively and a positive integer bound B.
> Question: Is there a node f of G that connects to each of the three nodes n_1, n_2, n_3 over a multi-hop path and such that the sum of the weighted hop-distance from each of the three nodes to f is no more than B?

Works [28][29][16] deal with the same subproblem and as a solution they use the closest node to the general Fermat point in Euclidian space [30]. This is a straightforward heuristic with an approximate solution. For this method to be efficient the network has to be dense, nodes need to have equal communication range and they need to be evenly distributed.

If those characteristics do not hold then the proposed heuristic can easily end up with a very bad choice for f as can be seen in Figure 9.

For our heuristic algorithm *2PH* the above proposed sub-optimal heuristic is good enough since it does not need to guarantee optimality and it is easy to implement. To find the general Fermat point in Euclidian space it takes constant time and using the a k-d tree we can find the closest node to the point in $O(logn)$ time. This replaces the exhaustive search for the optimal *Fermat* node that had a complexity of $O(n)$ and makes the time complexity of Phase 2 be just $O(logn * m)$. Thus the time complexity of algorithm *2PH* becomes $O(logn * m + m^2)$.

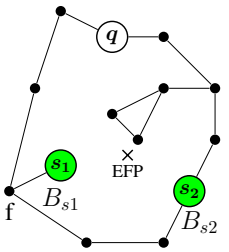

Fig. 9. The general Fermat Point for triangle uvq in Euclidian space is at the X mark denoted as EFP. The best node f to place the intersection operator however is not even close.

For our *DPopt* algorithm we need to use a heuristic that will guarantee optimality. We propose a method to find the optimal solution for the *GFPG* problem by cutting down drastically the initial search space of size n. Assuming that *Fermat node* is f, and $x = Hops(u, f)$, $y = Hops(v, f)$, $z = Hops(q, f)$ we use the following integer programming:

Given B_u, B_v, B_{uv}, $Hops(u, q)$, $Hops(u, v)$, $Hops(v, q)$ find x, y, z that minimize

$$B_u * x + B_v * y + B_{uv} * z$$

such that:

$$z + x \geq Hops(u, q),\ z + y \geq Hops(v, q),$$

$$x + y \geq Hops(u, v), \text{ where } x, y, z \text{ are integers}$$

$$x, y, z, B_i, Hops(i, j) > 0\ ,\ \forall i, j \in \{u, v, q\}$$

$$B_u*x+B_v*y+B_{uv}*z < B_u*Hops(u,v)+B_{uv}*Hops(v,q) \quad (7)$$

$$B_u*x+B_v*y+B_{uv}*z < B_v*Hops(u,v)+B_{uv}*Hops(u,q) \quad (8)$$

$$B_u*x+B_v*y+B_{uv}*z < B_u*Hops(u,q)+B_v*Hops(v,q) \quad (9)$$

For node f to be different from s_1,s_2,q it needs to satisfy inequalities 7,8 and 9. Assume a solution λ exists for the above integer programming problem thus there might be a node outside the sources and the sink that satisfies *GFPG*. All integer points P that lie in 3D space in the area formed by the hyperplanes of the constraints and the constraint $B_u * x + B_v * y + B_{uv} * z \geq \lambda$ are possible hop-distance combinations for our *Fermat* node f. For each $(x, y, z) \in P$ we try to find a node f in our network (graph) that is exactly x hops away from node u, y hops away from node v and z hops away from node q. This is performed by taking all x-hop neighbors of u, all y-hop neighbors from v and all z-hop neighbors from q. All nodes that satisfy this condition are candidate *Fermat* nodes. We now only need to search amongst those candidate nodes, instead of all the nodes in the network, to get the *Fermat node*.

Taking Figure 9 as an example with $B_{s1} = 10$, $B_{s2} = 10$ and $B_{s1s2} = 5$ then the solution for the integer programming is $x = 2, y = 2, z = 2$ with cost 50. All the integer (x, y, z) points in the area between the solution and the constraints are $(2, 2, 2), (2, 2, 3), (1, 3, 3), (3, 1, 3)$. In the network there is only one node f satisfying $x = Hops(u, f)$, $y = Hops(v, f)$, $z = Hops(q, f)$ for any of the above points, thus this is our *Fermat* node.

VIII. Hybrid algorithm

The proposed heuristic algorithm *2PHdeep* can be combined with the *DPopt* optimal algorithm to form a hybrid. We use the heuristic algorithm *2PHdeep* to produce a fixed operator sequence to consider when running the *DPopt* algorithm. This way *DPopt* does not need to compute the cost of all possible combinations of splits, but just of the fixed, predefined splits. The output then is the optimal operator placement for the given fixed operator sequence. In other words we run *Phase 1* of the *2PHdeep* algorithm and use the produced *evaluation tree* to cut down search space of *DPopt*. The solution will be suboptimal and the total complexity of the hybrid method is dominated by the complexity of the dynamic programming phase.

Lemma 2: The complexity of *Hybrid* is $O(m^2 * n^3)$, where m is the number of predicates in the query and n is the number of nodes in the network.

Proof: During the *Hybrid* algorithm the sets to be considered and their splits are already set by the $evaluationtree$. For each set size there are at most m sets in the tree and for each set there is only 1 split defined. This makes the $\sum_{k=1}^{\frac{i}{2}} \binom{i}{k}$ part of equation 3 become 1 and the $\binom{m}{i}$ part of the equation become m. Now the equation can be transformed into $\sum_{i=1}^{m} n^3 * m * 1$ and simplified into $m^2 * n^3$. ■

IX. Evaluation

First, we present two algorithms that will be used as comparison in our experiments.

A. Naive Algorithm

The routing algorithms for wireless sensor networks that have been proposed in literature are not sophisticated enough to exploit the characteristics of multi-predicate queries to reduce communication cost effectively. The best routing algorithms proposed in literature use opportunistic aggregation while streaming data to the sink [31] [32] [33]. This technique is also used for any type of aggregation query in the query optimizer of tinyDB [4]. First, the sink disseminates the query to all nodes of the network and during this process a communication (routing) tree is established (Figure 3). In the second phase, in a level-wise fashion, nodes send to their parents information which is then relayed to the level above, until all information reaches the sink. If a node receives two or more sets of values from its children or is a source, it computes the intersection of these sets before sending to its parent. As an example, consider the network of Figure 1 and the resulting

communication tree of Figure 3, established after the query has been disseminated to the network. First, nodes s_1, s_2 and s_3 transmit to their parents. s_2 intersects A_1 (received from s_1) with A_2 (i.e., its own set) to produce $A_1 \cap A_2$. This is sent to its parent q (the sink). n_4 relays A_3 to its parent and at the sink the final intersection is computed.

We will call this existing algorithm $Naive$. All proposed algorithms will be compared to $Naive$ which represents the state of the art technique used in WSNs to answer multi-predicate queries.

B. Element Exchange

Another method to answer MP-Queries in a network is to communicate elements instead of communicating whole lists. Based on this idea there are algorithms to minimize the needed number of comparisons in order to get the intersection [5]. We adapted the so called *Adaptive* algorithm presented in [8] to be run distributed by the source nodes inside the network.

In our implementation each source s_i that receives an element e_j from the previous source either forwards it if it is common with one of its own elements, or sends out the element e_k from its own list that has an immediate larger value than e_j. This is done cyclicly over all sources and whenever an element visits a source twice it is sent straight to the sink. This continues with the next larger element until a list runs out of elements. The task of the sink before running the query is to define the order in which the sources will communicate in a circle.

We will call this algorithm *ElEx* and we order the sources by picking a random source to start with and then picking as a successor the closest one from the remainding in a greedy fashion.

C. Experiments

Experiments were run on a network simulator in C++. We used two network layouts: a uniform grid with varying number of nodes from 10 to 150 and a random network with 150 randomly placed nodes in a space of 1000x1000 with communication radius of 125. The first experiments were run without any objects, only considering list sizes and a constant selectivity as parameter for the intersections, in order to see the impact of the intersection selectivity on our algorithms. This selectivity mostly depends on the size of the smallest list taking part in the intersection, but also on the number of all lists involved in the intersection and their sizes. For further experiments we used two datasets. One dataset has 1000 objects and for each object random points are uniformly generated over the whole network space and stored on appropriate nodes. Datapoints are generated until each node contains approximately half the objects in order to avoid early empty intersection which would favor our algorithms. The second dataset is generated using a network-based data generator for moving objects [34] in order to evaluate the algorithms in the case of a realistic dataset with spatial data correlation. We used the Oldenburg road map and 8000 object to be generated. Their movement simulated for 1000 timesteps for a total of 200000 points in space. Random queries were generated by randomly picking a sink node and sources with varying number of sources m.

The efficiency of each algorithm is expressed as the percentage of the cost needed by the *Naive* algorithm described above (IX-A). In our simulator the energy dissipated for communication is based on the models presented in [35] with the transmission range set constant. In our simple simulator fixed costs such as message headers are not considered since our cost model is not targeted to any particular sensor device or network protocol. Thus, the number of elements sent over a link is used to calculate cost units. Ignoring the packet overhead greatly favors the element exchange algorithm *ElEx* since a large amount of packets are used to send little data (1 element per packet). All measurements are averaged over 20 random queries for each different parameter: network size n, predicate number m, selectivity $selec$, and dataset used. Table III summarizes the notations we will use for each individual algorithm. Note that due to its high complexity *DPopt* could not be run for predicates more that 8.

TABLE III
ALGORITHMS TO BE COMPARED

2PH	The *2PH* algorithm presented in Section VI
2PHdeep	The *2PH* algorithm that implements the optimization proposed in Section VI-A
Hybrid	Algorithm that combines the evaluation tree returned from *2PHdeep* and the Brute Force operator placement given by *DPopt*
DPopt	The optimal solution as presented in Section V
ElEx	The element exchange algorithm (IX-B)

Impact of intersection selectivity: The gain over the *Naive* algorithm becomes smaller as the intersections get less selective. The *selectivity* value can be translated as multiplying the smallest input list with it in order to get the size of the intersection result. Thus, bigger values mean less selective intersections. We can see in Figure 10 that the less selective the intersection the smaller the gain compared to *Naive*. For the constant selectivity experiments the *ElEx* algorithm could not be run because no real objects were used in this experiments, thus no elements to exchange. The calculation of the cost was based on list sizes, which were originally assigned in random to the sources, and the selectivity factor for the intersections.

Impact of query size m: The more predicates we have in the query the larger the improvement compared to *Naive*, as shown in Figures 11, 12, 13. This is expected since the more predicates the more lists need to be intersected and thus the more data will be reduced before reaching the sink. Note that the intersections might even end up in an empty result, in which case we send an "empty"-element through the rest of the operator nodes and the sink to convey the empty result. For the experiments run on the random dataset where no spatial correlation in the data exists (Figure 12) we can also see the performance of *ElEx*. For small number of predicates it performs even worse than *Naive*. This is because sources are highly probable to be far apart and need to exchange many

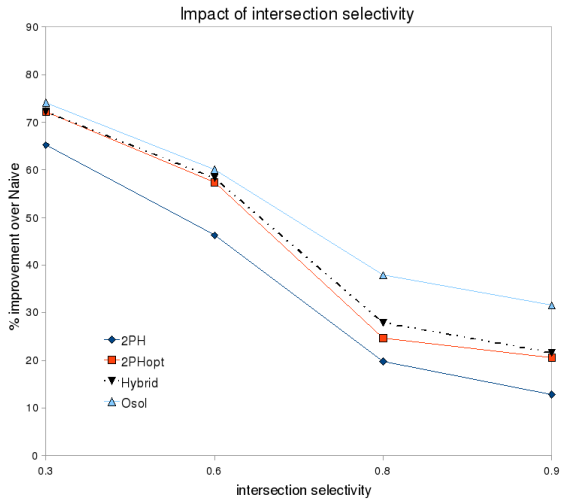

Fig. 10. Experiments no objects were used and selectivity was constant

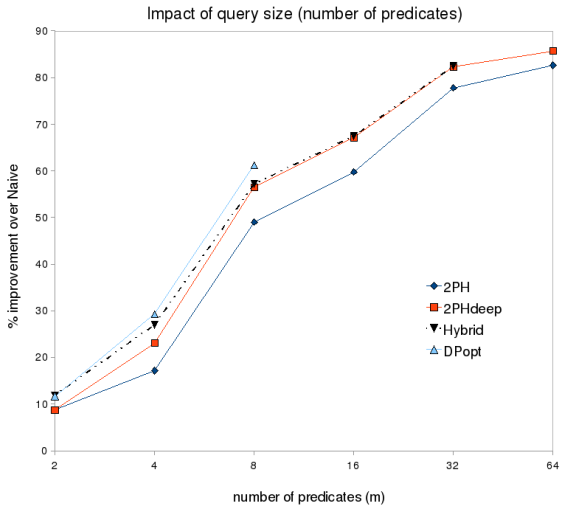

Fig. 11. Experiments no objects were used and selectivity was constant. The selectivity factor has values ranging from .3 to .9

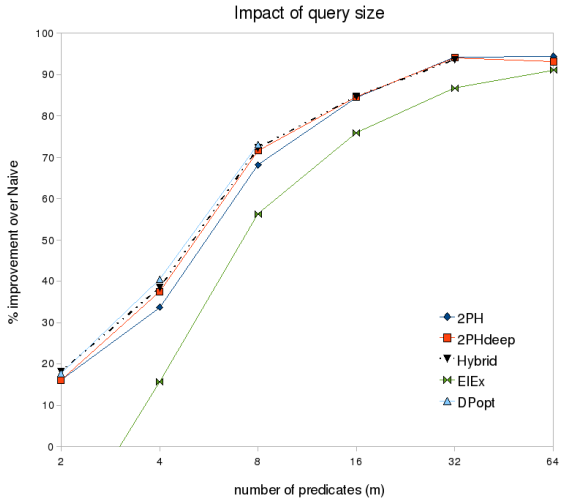

Fig. 12. Experiments with randomly generated data points.

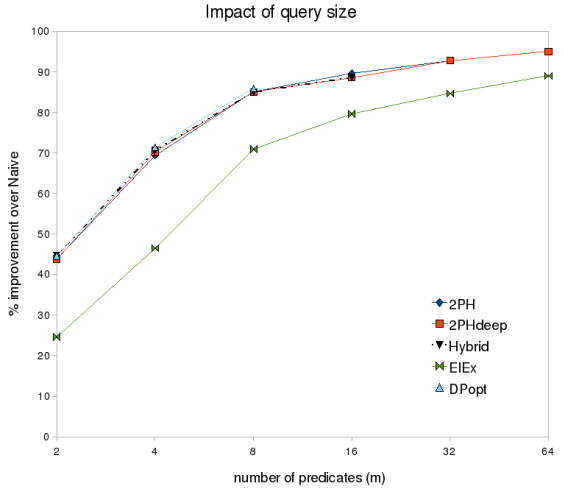

Fig. 13. Experiments with spatially correlated data points.

elements in order to reach the intersecting result. With more sources (predicates) the distance between sources is bound to be smaller thus resulting in more "local" element exchange. For the correlated dataset (Figure 13) the performance of *ElEx* is better. Even when for small number of sources the data has to travel further in order to intersect, when it does so, the intersection is much smaller due to the spatial correlation of the data: the further away you go the less common elements you have. This is why for larger values of m the cost of evaluating a query compared to the *Naive* cost is very low because there are many lists to intersect and the probability of non-common objects is higher.

The heuristic algorithm *2PHdeep* performs always better than the algorithm used so far (*Naive*). It always outperforms the element exchange algorithm *ElEx* even if the latter is favored by the cost function as mentioned earlier. Even more notable is the fact that it performs always very close to the optimal solution (*DPopt*), showing robustness to the selectivity and the correlation of the data, to the size of the queries and the network.

X. Summary and Conclusion

Recent research has focused on processing joins in wireless sensor networks. None so far has tackled the problem of in-network processing of queries with multiple intersections in a holistic manner. We show that the problem of minimizing communication cost of such queries is NP-hard and develop a dynamic programming algorithm together with a heuristic to compute the optimal solution for small problem instances. The heuristic used is based on a variation of the general Fermat point problem which is for the first time addressed and solved here. We also propose a much faster sub-optimal algorithm

that is almost as efficient as the optimal.

The extensive experimental evaluation compares the proposed algorithms to the most widely used technique used to evaluate queries in wireless sensor networks and shows that an improvement of 10% to 95% is possible. The low complexity heuristic algorithm is also shown to be scalable and robust to different query characteristics and network size. Also, it is straightforwardly implementable into the optimizer of TinyDB [4]. The proposed algorithms, can be applied to any high communication cost network where there is a need to combine (intersect) data from different sources.

Acknoledgments: This work was partially supported by grants NSF IIS-0534781, NSF 0803410, ONR N00014-07-C-0311 AWARE project, and by grant HKU 7155/06E from Hong Kong RGC

REFERENCES

[1] A. Banerjee, A. Mitra, W. Najjar, D. Zeinalipour-Yazti, V. Kalogeraki, and D. Gunopulos, "Rise - co-s : high performance sensor storage and co-processing architecture," *Sensor and Ad Hoc Communications and Networks, 2005. IEEE SECON 2005. 2005 Second Annual IEEE Communications Society Conference on*, pp. 1–12, Sept., 2005.

[2] D. Zeinalipour-Yazti, V. Kalogeraki, D. Gunopulos, and W. A. Najjar, "Data acquisition in sensor networks with large memories," in *ICDE Workshops*, 2005, p. 1188.

[3] M. Hadjieleftheriou, G. Kollios, P. Bakalov, and V. J. Tsotras, "Complex spatio-temporal pattern queries," in *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*. ACM, 2005, pp. 877–888.

[4] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tinydb: an acquisitional query processing system for sensor networks," *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 122–173, 2005.

[5] E. D. Demaine, A. López-Ortiz, and J. I. Munro, "Adaptive set intersections, unions, and differences," in *SODA '00: Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2000, pp. 743–752.

[6] J. Barbay and C. Kenyon, "Adaptive intersection and t-threshold problems," in *SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2002, pp. 390–399.

[7] R. A. Baeza-Yates, "A fast set intersection algorithm for sorted sequences," in *Combinatorial Pattern Matching, 15th Annual Symposium, CPM 2004, Istanbul,Turkey, July 5-7, 2004, Proceedings*, ser. Lecture Notes in Computer Science, vol. 3109. Springer, 2004, pp. 400–408.

[8] J. Barbay, A. Lpez-Ortiz, and T. Lu, "Faster adaptive set intersections for text searching," pp. 146–157, 2006. [Online]. Available: http://www.springerlink.com/content/7w4424l1m318000

[9] H. Lu and M. J. Carey, "Some experimental results on distributed join algorithms in a local network," in *11th International Conference on Very Large Data Bases*. Stockholm, Sweden: Morgan Kaufmann, 21–23 Aug. 1985, pp. 292–304.

[10] L. F. Mackert and G. M. Lohman, "R* optimizer validation and performance evaluation for distributed queries," in *Twelfth International Conference on Very Large Data Bases*. Kyoto, Japan: Morgan Kaufmann, 25–28 Aug. 1986, pp. 149–159.

[11] J.-M. Chang, "A heuristic approach to distributed query processing," in *VLDB '82: Proceedings of the 8th International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1982, pp. 54–61.

[12] M.-S. Chen and P. S. Yu, "A graph theoretical approach to determine a join reducer sequence in distributed query processing," *IEEE Transactions in Knowledge Data Engineering*, vol. 6, no. 1, pp. 152–165, 1994.

[13] J. M. Hellerstein, "Optimization techniques for queries with expensive methods," *ACM Trans. Database Syst.*, vol. 23, no. 2, pp. 113–157, 1998.

[14] Monma, Clyde L. and Sidney, Jeffrey B., "Sequencing with series-parallel precedence constraints," *Mathematics of Operations Research*, vol. 4, no. 3, pp. 215–224, aug 1979.

[15] A. Meliou, D. Chu, J. M. Hellerstein, C. Guestrin, and W. Hong, "Data gathering tours in sensor networks," in *Proceedings of the Fifth International Conference on Information Processing in Sensor Networks, IPSN 2006, Nashville, Tennessee, USA, April 19-21, 2006*. ACM, 2006, pp. 43–50.

[16] A. Coman, M. A. Nascimento, and J. Sander, "On join location in sensor networks," in *MDM*, 2007, pp. 190–197.

[17] H. Yu, E.-P. Lim, and J. Zhang, "On in-network synopsis join processing for sensor networks," in *MDM '06: Proceedings of the 7th International Conference on Mobile Data Management*. Washington, DC, USA: IEEE Computer Society, 2006, p. 32.

[18] T.-W. Chen and M. Gerla, "Global state routing: a new routing scheme for ad-hoc wireless networks," *Communications, 1998. IC 98. Conference Record*, vol. 1, pp. 171–175, 7-11 Jun 1998.

[19] B. Karp, "Gpsr: Greedy perimeter stateless routing for wireless networks," pp. 243–254, 2000.

[20] S. Lin, B. Arai, and D. Gunopulos, "Reliable hierarchical data storage in sensor networks," *Scientific and Statistical Database Management, 2007. SSBDM '07. 19th International Conference on*, pp. 26–26, July 2007.

[21] D. Zeinalipour-yazti, S. Lin, V. Kalogeraki, D. Gunopulos, and W. A. Najjar, "Microhash: An efficient index structure for flash-based sensor devices," in *In FAST*, 2005, pp. 31–44.

[22] C. Estan and J. Naughton, "End-biased samples for join cardinality estimation," *Data Engineering, 2006. ICDE '06. Proceedings of the 22nd International Conference on*, pp. 20–20, April 2006.

[23] R. J. Lipton, J. F. Naughton, and D. A. Schneider, "Practical selectivity estimation through adaptive sampling," 1990, pp. 1–11.

[24] Y. Tao, G. Kollios, J. Considine, F. Li, and D. Papadias, "Spatio-temporal aggregation using sketches," in *ICDE*. IEEE Computer Society, 2004, pp. 214–226.

[25] B. H. Bloom, "Space/time tradeoffs in hash coding with allowable errors," *Comm. of the ACM*, vol. 13, no. 7, p. 422, July 1970.

[26] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*. NY: Plenum Press, 1972, pp. 85–103.

[27] S. Guha, R. Rastogi, and K. Shim, "Cure: an efficient clustering algorithm for large databases," *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pp. 73–84, 1998.

[28] Y.-M. Song, S.-H. Lee, and Y.-B. Ko, "Ferma: An efficient geocasting protocol for wireless sensor networks with multiple target regions," in *EUC Workshops*, 2005, pp. 1138–1147.

[29] J. Son, J. Pak, H. Kim, and K. Han, "A decentralized hierarchical aggregation scheme using fermat points in wireless sensor networks," in *EvoWorkshops*, 2007, pp. 153–160.

[30] R. Courant and H. Robbins, *What Is Mathematics?* Oxford University Press, 1941.

[31] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: a scalable and robust communication paradigm for sensor networks," in *MOBICOM*, 2000, pp. 56–67.

[32] B. Krishnamachari, D. Estrin, and S. B. Wicker, "The impact of data aggregation in wireless sensor networks," in *ICDCS Workshops*. IEEE Computer Society, 2002, pp. 575–578.

[33] S. Madden, M. J. Franklin, J. Hellerstein, and W. Hong, "Tag: a tiny aggregation service for ad-hoc sensor networks," in *In OSDI*, 2002.

[34] T. Brinkhoff, "A framework for generating network-based moving objects," 2002.

[35] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient communication protocol for wireless microsensor networks," in *Hawaii International Conference on System Sciences*, 2000. [Online]. Available: http://computer.org/proceedings/hicss/0493/04938/04938020abs.htm