Multiresolution Cube Estimators for Sensor Network Aggregate Queries

Alexandra Meliou[1], Carlos Guestrin[2], and Joseph M. Hellerstein[3]

[1] University of Washington ameli@cs.washington.edu**
[2] Carnegie Mellon University guestrin@cs.cmu.edu
[3] University of California Berkeley hellerstein@cs.berkeley.edu

Abstract. In this work we present in-network techniques to improve the efficiency of spatial aggregate queries. Such queries are very common in a sensornet setting, demanding more targeted techniques for their handling. Our approach constructs and maintains multi-resolution cube hierarchies inside the network, which can be constructed in a distributed fashion. In case of failures, recovery can also be performed with in-network decisions. In this paper we demonstrate how in-network cube hierarchies can be used to summarize sensor data, and how they can be exploited to improve the efficiency of spatial aggregate queries. We show that query plans over our cube summaries can be computed in polynomial time, and we present a PTIME algorithm that selects the minimum number of data requests that can compute the answer to a spatial query. We further extend our algorithm to handle optimization over multiple queries, which can also be done in polynomial time. We discuss enriching cube hierarchies with extra summary information, and present an algorithm for distributed cube construction. Finally we investigate node and area failures, and algorithms to recover query results.

1 Introduction

Sensing devices are now used in many practical applications that require monitoring of physical phenomena. The data generated by such applications poses new challenges to data management research, prompting recent research on wireless sensor networks to devote significant attention to query processing [1]. Model-based data acquisition schemes [2] use historical information to predict rough query answers from probabilistic models, and optimize node selection to meet the desired query accuracy. Various centralized approaches [3, 4] aim to optimize communication strategies for data gathering, assuming accuracy of a centrally maintained model, which makes them more prone to failures.

Other query specific methods promote in-network modeling and decisions [5], but they are geared towards general `SELECT *` type queries that request values at different sensor locations. In this work we address *aggregate* queries, i.e. queries that request a certain type of summary information over a group of sensor nodes. Their differences from `SELECT *` queries require us to explore new approaches for the optimization of aggregates. Since sensornets are commonly used for monitoring physical phenomena over a specific area, such aggregates are usually spatially constrained, e.g. what is the average temperature in the engine room. *Spatial interest queries* define a region of interest as a selection criterion, and query results are computed based on datapoints within that region.

** This work was conducted while affiliated with University of California Berkeley.

Localized interest naturally lends itself to in-network summarization schemes, like the hierarchical summaries used in [5]. In this paper we propose *multiresolution cube hierarchies* as a way to efficiently summarize and query spatially restricted aggregate data. This focus is well suited to the sensornet setting due to the type of applications that these environments usually support. Our scheme generalizes data cubes [6] to represent area aggregates at different resolutions over the network. The cube hierarchies store information at different granularities (resolutions) allowing them to be applicable to queries of various ranges of region sizes. Detailed description of the data stored in the network cube hierarchies is given in Section 2.

We further demonstrate how the multiresolution cubes can be used to efficiently answer spatial aggregate queries. Section 3 presents a polynomial algorithm that selects the minimum number of datapoints required to construct the answer for a specific query, and Section 3.1 extends it to the case of multi-query optimization. Section 4 discusses an alternative method of summarization that maintains richer summaries, and presents an algorithm that performs distributed construction of the cube hierarchies.

Multiresolution cubes are effective against node failures, which commonly occur in sensor network settings. Section 5 discusses isolated node failures as well as area outages, and demonstrates how the missing data can be recovered from other locations in the cube. Finally, Section 6 discusses related work and Section 7 touches on future directions.

2 Multiresolution Network Cubes

We focus on a type of in-network data summaries that will provide a framework to efficiently respond to aggregate queries with spatial constraints. Such queries are common in many applications, and are of the form `SELECT avg(temperature) WHERE nodeID inRegion [(2,3),(5,9)]`. We target distributive (e.g. SUM) and algebraic aggregates (e.g. AVG), and assume queries that specify arbitrary rectilinear planar regions as their selection criterion. We do not address holistic aggregates in this work. Regions of interest are defined as a set of points on the plane, and can be generalized to 3 dimensions, but in this paper we focus on the 2-dimensional case. In the simplest case, the region of interest is rectangular, and can be defined by just 2 corners, e.g. upper left and lower right. Regions can be of arbitrary shape, but those can always be split into simpler rectangles. The corner points of the interest areas can be given as coordinates in the euclidean space.

For the purposes of this paper we will assume sensing locations arranged on a 2-dimensional grid, and regions of interest defined over this grid. Grid topologies have been studied before in the literature [7, 8]. Note however that this assumption does not restrict the application domain to grid deployments. Grid locations do not need to correspond to actual sensor locations, as the grid can be an overlay over the actual network topology, which we briefly discuss in Section 7.

Queries define an area of interest over the grid specified by the locations of corner points, and request the computation of an aggregate value over the specified region. An example is shown in Figure 1. Containment of a grid location within the area of interest can be easily determined based on the region's point coordinates. Without loss of generality we will from now on focus on SUM as the aggregate function, but our scheme can be easily adapted to other distributive and algebraic aggregates, which can be expressed as a scalar function of distributive functions.

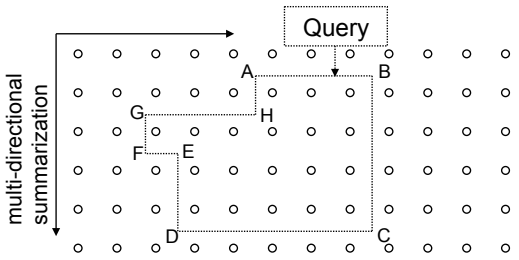


Fig. 1. Spatial queries can define arbitrary rectilinear regions over the grid locations.

In order to facilitate aggregate computation inside the interest region, we aim to preserve partial summary information within the network. The grid provides a natural setting for constructing cube summaries, as those can be computed along the grid directions, as in traditional data cubes. Without loss of generality we can assume that the top left corner of the grid is location $(0, 0)$, with coordinates increasing from left to right and top to bottom, and that sum information is maintained along the increasing directions. These localized summaries can be stored at the corresponding grid locations and used to answer spatial queries without requiring the retrieval of data from all the corresponding nodes.

In order to accommodate queries of various region sizes and shapes, summarization can be performed at multiple granularity levels. The grid can be simply divided into rectangular cells of equal size, forming *level-1* cubes where local summaries can be computed independently. The level-1 cubes cover the grid in a non-overlapping fashion and each summary is stored at a pre-specified cell location – without loss of generality for this discussion we will assume that this location is the lower right corner of each cell.

Level-1 cells will now be treated as the new unit entities and get grouped to form level-2 cells. The process is repeated across multiple levels leading to a structure resembling a quad-tree. A depiction of a simple two level hierarchy can be seen in Figure 2. In this example a level-1 cube cell contains the sum of the data from a 3×3 area of the grid, and a level-2 cell sums over four level-1 cells. Note that a specific grid location may contain the sum data of more than one cell-level. Due to the process of their construction, multiresolution cubes have the following properties: (a) cube cells of the same level do not intersect, (b) if cell A of level i, and cell B of level $j < i$ intersect, then $B \subset A$.

2.1 Mapping Query Regions to Cube Cells

Cube summaries are stored in a distributed fashion in the network, at a specified location within the cell area that generated them, and can be used to compute aggregates over

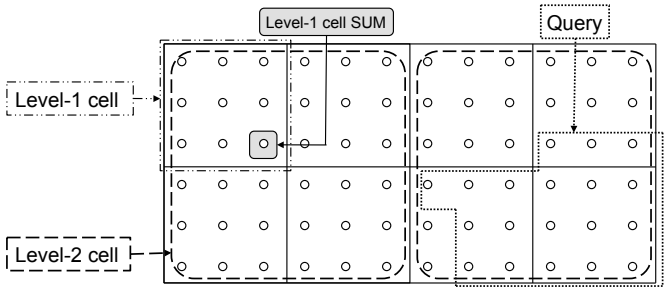


Fig. 2. A two level cube hierarchy. Queries can span cells at different granularities.

query specified regions without the need to access all the corresponding grid location. Query regions can be of arbitrary shape, spanning cells of different levels in the cube hierarchy, and our goal is to select the smallest number of cells that can reconstruct the requested aggregate. Since cube cells of the same level do not overlap, it is always most efficient to use the higher level cells that are contained in a region. An arbitrary query region can be decomposed into smaller rectangles based on the multi-resolution cube hierarchy. Then the region of interest becomes a collection of non-overlapping cells, whose data can be used to compute the aggregate over the whole region. Since cells of the same hierarchy level are not overlapping, the query area can be greedily split into cells in an optimal way, minimizing the number of cells that comprise it.

Algorithm 1 gives a sketch of the greedy division of a region into hierarchy cells. In the algorithm description, a corner of the interest region is *convex* if the region is locally convex in the immediate area surrounding the corner. For example, in Figure 1 the corners A, B, C, D, F, G of the query region are convex, while E and H are concave.

Lemma 1. [4] *Algorithm 1 produces the minimum number hierarchy cells that exactly cover a given query region.*

Algorithm 1 Greedy Region Division (pseudocode)

1: **repeat**
2: Select a *convex corner* c of the region of interest G.
3: Select $\max_k C_k$ (C_k cell of level k) such that $C_k \subseteq G$ and $c \in C_k$.
4: Extract C_k from G.
5: **until** $G = \emptyset$

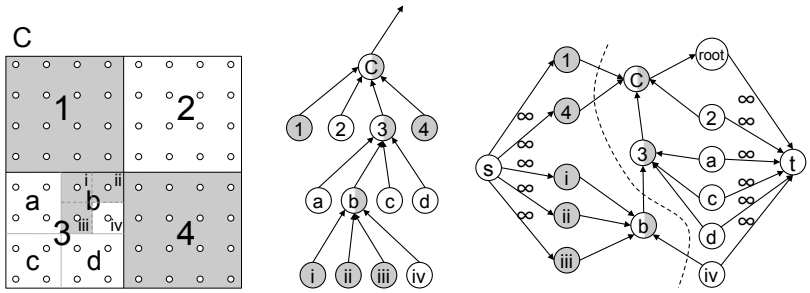
3 Optimization of Spatial Aggregate Queries

In the previous section we showed how a query region can be greedily mapped onto hierarchy cells, so that the minimum number of cells are used to cover the region. Algorithm 1 provides the optimal mapping of a query region to hierarchy cells, but this may not be the optimal strategy to compute a query aggregate.

In the example of Figure 3a cube C is further divided into 3 resolution levels, with some of the cell divisions displayed. Assume a spatial aggregate query over the shaded area G. Algorithm 1 can map G to the cube cells $S = 1, 4, i, ii, iii$. The query aggregate can then obviously be computed using the summary values of cells in S: $V(G) = V(1) + V(4) + V(i) + V(ii) + V(iii)$, where $V(x)$ refers to the aggregate value (sum) of cell or region x. The set S does not however provide a unique solution to the problem of aggregate reconstruction. $V(G)$ can also be computed as $V(C) - V(2) - V(a) - V(c) - V(d) - V(iv)$, and there are numerous other possibilities. Most importantly, note that S does not provide the minimum solution either, in regards to the number of data points that need to be retrieved. In this example the smallest set of data points that can reconstruct $V(G)$ is $1, 4, b, iv$: $V(G) = V(1) + V(4) + V(b) - V(iv)$.

The problem that we want to solve is the following: given a multi-resolution cube hierarchy and a spatial aggregate query, select the minimum number of data points (cube aggregates) that are needed to compute the query answer. With this optimality criterion,

[4] Proofs omitted due to space constraints.



(a) (b) (c)

Fig. 3. (a) The grey area depicts the area of interest of a query $G = \{1, 4, i, ii, iii\}$ over the grid . (b) Multiresolution cube as a tree hierarchy. (c) Transformation to a max-flow problem

the optimal query plan for a spatial aggregate query can be computed in polynomial time, through a polynomial reduction to a max flow problem. We will demonstrate the methodology with the example of Figure 3a.

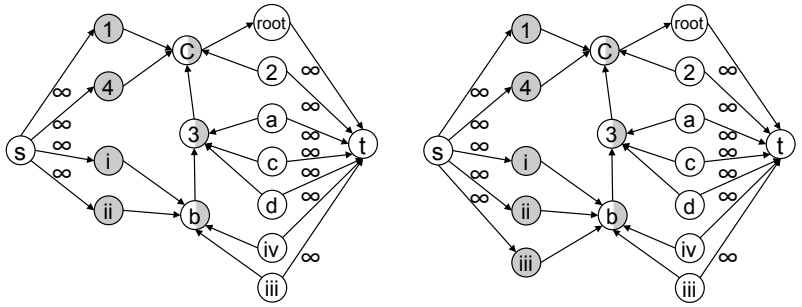
A multi-resolution cube can be represented as a tree hierarchy, where every node represents a cube cell that contains all the cells in its subtree. Figure 3b shows the tree representation of the example from Figure 3a. The grey-shaded nodes are cube cells completely contained in the query region G, white nodes do not overlap with G, and partially shaded nodes only partially overlap with G. Note that descendants of completely shaded or completely unshaded nodes are omitted, as their inclusion in the result would be sub-optimal since a fully grey or fully white cell always dominates them. Edge direction simply denotes containment of cells across the different resolution levels.

The hierarchy can transformed into a flow problem as shown in Figure 3c. A source node s is connected with infinite capacity edges to all the fully shaded nodes, and all white nodes with the inclusion of a root node as parent of the entire hierarchy are connected to a target node t, also with infinite capacity edges. The remaining edges are assigned unit capacity. This transformation conceptually partitions the nodes into those whose summary positively contributes to the query aggregate as they are contained in the interest region (shaded nodes), and those whose summary should not be included in the aggregate computation as they are outside the interest region (white nodes). Note that a *cut* in the graph in Figure 3c simply assigns the partially shaded nodes into one of the two partitions. If a semi-shaded node is assigned to the grey partition (e.g b), then appropriate white nodes should be subtracted from the SUM (e.g. iv). Given an $s-t$ cut in the transformation graph with edge set E that partitions the nodes into sets S and T, with $s \in S$ and $t \in T$, then the total sum over the interest region is computed as

$$V(G) = \sum_{\substack{v \in S, u \in T \\ (v,u) \in E}} V(v) - \sum_{\substack{v \in S, u \in T \\ (u,v) \in E}} V(u)$$

The cut depicted in Figure 3c is actually the min-cut solution of size 4 for this graph, and corresponds to the summation $V(G) = V(1) + V(4) + V(b) - V(iv)$.

Theorem 1. *The minimum number of data points from a multi-resolution cube hierarchy that are sufficient to answer a spatially constrained aggregate query is equal to the minimum $s-t$ cut in the corresponding flow graph.*



(a) (b)

Fig. 4. (a) Flow graph for query Q_2 (b) Combined flow graph for queries Q_1 and Q_2

Theorem 1 provides a correspondence to a PTIME algorithm for computing the optimal set of data points in a cube hierarchy that can compute the answer to a spatial aggregate query. Using for example the Ford-Fulkerson algorithm to solve the corresponding max-flow problem we can select the optimal set of data points in $O(nf)$ where f the maximum graph flow. In our case f is bounded by n which corresponds to the number of data points.

3.1 Dealing with Multiple Queries

In the previous section we investigated optimal aggregate computation for a given spatially constrained query. However, when multiple queries run simultaneously in the system, optimizing them individually is not guaranteed to yield the overall optimal result.

Consider the cube hierarchy of Figure 3a, and Q_1 a query with interest region the one depicted in the figure: $G^{Q_1} = \{1, 4, i, ii, iii\}$. Also assume query Q_2 for which $G^{Q_2} = \{1, 4, i, ii\}$. The queries are mostly overlapping, apart from cell iii which is included in Q_1 but excluded in Q_2. The optimal set of data points that computes Q_1 as seen in the previous section is $S_1 = \{1, 4, b, iv\}$. Similarly, computing the optimal set for Q_2 using the flow graph of Figure 4a yields set $S_2 = \{1, 4, i, ii\}$. This approach requires $|S_1 \cup S_2| = 6$ data points to answer both queries. Note however that this is not the best solution as we can compute both query results by retrieving just 5 elements: $\{1, 4, i, ii, iii\}$.

The solution is to use a *combined flow graph* by appropriately merging the individual flow graphs of all the queries we need to optimize for. In rough terms, the combined flow graph is a union of the individual graphs, creating replicas for nodes appearing in different groups (shaded/white/undetermined) in the base graphs. An example is given in Figure 4b, which is a combination of the graphs in Figures 3c and 4a. Note that in this example node iii is shaded in one graph and white in the other, resulting in two instances, one white and one shaded, in the combined graph. Each of the original graphs is just a subgraph of the combined flow graph.

Proposition 1. *Two edges in the combined flow graph that correspond to the same data point cannot be part of the same cut.*

The combined graph contains all the information of the original graphs. A cut in the combined graph defines cuts in the original graphs, and therefore a cut in the combined graph corresponds to solutions for all queries that created it.

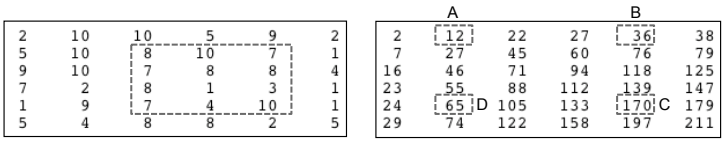


Fig. 5. Example of Prefix-Sum. The sum within the denoted region can be computed from 4 data points: $170 + 12 - 36 - 65 = 81$

4 Prefix-Sum Cubes

In this section we will examine an alternative summarization scheme, which can enrich multi-resolution cubes with additional summary information. Up to this point we assumed a basic summarization approach that maintains the total sum of elements in each cube-cell. This method may under-utilize some of the grid locations, as cell summaries are kept in a subset of the nodes.

A simple augmentation of the cell summaries that can better utilize the grid structure is the application of the *prefix-sum* algorithm [9]. Prefix sum (PS) works on the grid by storing at location (i^*, j^*) the sum of the values from all locations where $i \leq i^*$ and $j \leq j^*$. Figure 5 displays an example of the prefix sum algorithm: the first matrix contains the individual data, representing the values at each grid location, and the second matrix shows the result of the prefix-summation.

PS values are stored in all nodes instead of just one node per cell. Again we can build multi-resolution hierarchies, by splitting the grid into cells at the lowest level, performing PS at each one, and reiterating at the next level with each cell as a new base element. An advantage of using prefix-sum is that it allows for finer granularities in the aggregate computation. Even a single level cube can allow the computation of the sum within a rectangular region using just the four corner points due to the way prefix-sums are computed. In the example of Figure 5 the region sum is equal to $C + A - B - D$.

Proposition 2. *The total number of data points needed to calculate the sum within an arbitrary rectilinear region in a prefix-sum cube is the same as the number of its corners.*

Cubes with PS information have more storage requirements, but can result in more efficient plans (fewer data points that need to be retrieved). A $k \times k$ cube-cell stores $2k^2 - 1$ different sums over the finer resolution level. Each of these data points refers to a rectangular region with upper left corner the upper left corner of the cell, and lower right corner the grid point (i, j) where this particular data point is stored. The data points are divided into three sets based on whether they overlap with the query region G: $\forall s_i \in S_g$ $\{s_i\} \cup G = G$, $\forall s_i \in S_w$ $\{s_i\} \cap G = \emptyset$, and $\forall s_i \in S_u$ $\{s_i\} \cap G \neq \emptyset$ and $\{s_i\} \cup G \supset G$.

We construct a "re-colored" set S_c from S_u as follows: $\forall s_i \in S_u$ we select a subset $S'_w \subset S_w$ such that $s'_i = \{s_i\} \setminus \bigcup_j \{s_j \in S'_w\}$, $\{s'_i\} \cup G = G$ and $(\{s_i\} \setminus \{s'_i\}) \cap G = \emptyset$. We also assign $cost(s'_i) = |S'_w| + 1$. Finally $S_c = \bigcup_i s'_i$. We can now select the data points from the prefix sum cube that reconstruct the query with dynamic programming, using Algorithm 2, where G is the grey area of interest, $S = S_g + S_c$ and $c = 0$.

4.1 Distributed Construction of Multi-resolution Cube Hierarchies

Construction of the cube should be done in a distributed fashion, and should not be communication intensive. In this section we will present a distributed algorithm for the cube

Algorithm 2 PSQuery(G,S,c)

```
1: if G = ∅ or S = ∅ then
2:    return c
3: end if
4: for s in S do
5:    i=i+1
6:    G'=G-s
7:    S'={s ∈ S s.t. s contained in G}
8:    c'=c+cost(s)
9:    A(i)=PSQuery(G',S',c')
10: end for
11: return min(A(i))
```

construction, which requires only a single transmission by every node. Our algorithm is designed to support the construction of prefix-sum hierarchies, but the additional prefix-sum values can be simply dropped to revert to the simple cube scheme discussed in Sec. 2.

We assume that nodes know their location on the grid, and that a single packet has enough space for h values, where h is the wanted height of the hierarchy. Every node will store up to h values, depending on how many levels of the hierarchy it participates in.

Given a set of fanouts $\{F_i\}$ for the various hierarchy levels (number of cells that comprise higher level cells), a node can tell if it serves as a *junction* for level k, i.e. the node that stores the sum of the current level-k cell, if both its coordinates are divisible by $\prod_1^k F_i$. If a node is a junction for level k, it needs to forward its k-level sum to level $k+1$, but nothing to levels $> k+1$. A node adds its $k-1$ value, to the k-level message, thus adding to the k-level sum. Also, a node knows based on its coordinates if it lies at the border of a level region. The specifics are given in Algorithm 3. With this scheme, every node broadcasts one packet, and receives 3. This allows the algorithm to scale very well, as the number of message transmissions and receipts is independent of the network size.

Algorithm 3 Distributed Cube Construction (algorithm description)

1: A node with coordinates (x,y) expects messages from nodes (x,y-1), (x-1,y) and (x-1,y-1). If one or more of these nodes don't exist ((x,y) is at the edge of the grid), then (x,y) proceeds without those messages.
2: A packet is of the following form: $P = [F_1 : value, F_2 : value, \ldots, F_h : value]$.
3: F_0 is defined as the local value at every node.
4: Every node will store $k+1$ values, where k is the level for which the node is a junction. By default all nodes are junctions for level 0.
5: Upon receipt of the 3 packets P_a from (x,y-1), P_b from (x-1,y) and P_c from (x-1,y-1), it computes $P(F_i) = P_a(F_i) + P_b(F_i) - P_c(F_i)$.
6: If x-1 is divisible by $\prod_1^k F_i$, then node (x,y) sets $P_b(F_k) = P_c(F_k) = 0$ before making the above computation. If y-1 is divisible by $\prod_1^k F_i$, then node (x,y) sets $P_a(F_k) = P_c(F_k) = 0$ before making the above computation.
7: A k-level junction node stores level values up to level $k+1$, where for level i the value is $P(F_i) + local(F_{i-1})$.
8: The node builds a new packet P with the $k+1$ values that it has stored. It also populates it with the values $P(F_{k+2}), \ldots P(F_h)$ as computed upon receipt.

5 Handling Failures

In the previous sections we discussed the structure and distributed construction of multi-resolution cube hierarchies, either with simple or prefix-sum summarization schemes,

and developed algorithms that use these in-network estimators to efficiently answer spatially constrained queries. In this section we will discuss data recovery after node failures, which is a common occurrence in sensornet deployments. Our multi-resolution cubes implement data redundancy, making full recovery feasible in many cases of failures. Especially in the case of prefix-sum hierarchies, most node values can be reconstructed by simply querying 3 immediate neighbors.

The k-level value for a specific node is constructed as $local(F_i) = P_a(F_i) + P_b(F_i) - P_c(F_i) + local(F_{i-1})$ (see Algorithm 3). Therefore, any node that is used as an a,b or c-node in this equation (locations $(x, y-1)$, $(x-1, y)$ and $(x-1, y-1)$ respectively) can have its value reconstructed by querying the other 3 nodes in the local square. The only complication arises with junction nodes. A junction node does propagate its value in the same fashion, but the receiving nodes do not store it but only forward it until it reaches the appropriate node for that hierarchy level. That means that if a node that serves as a junction for level i fails, then we need to access data at a total distance of $3F_i$ (F_i the current fan-out) from that node to reconstruct the missing value.

Another alternative would be to store at every node up to level $k+2$ values instead of $k+1$. The construction scheme would not change, as that information is already sent around the network, but nodes would be required to store one extra value. That would allow the reconstruction of the value after a single failure with just querying the 3 immediate neighbors. This implies a tradeoff between storage space and recovery capability, which can be further investigated in future work.

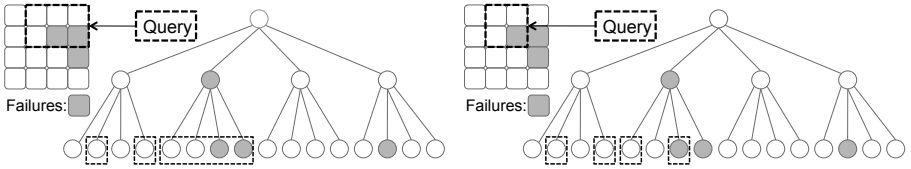
An important point to note is that the recovery logic is tied to the method of construction, which is performed with local planning and decisions. The location of data points that can reconstruct missing measurements can be determined locally, and therefore upon detection of failures, recovery can be performed with in-network decisions.

5.1 Area Failures

Isolated node failures can be resolved in a straightforward fashion using the values of the neighbors of the failing nodes. Area failures introduce more complications, but they can still be addressed, and fully reconstructed in many cases. We identify area failures as those that affect an arbitrary number of consecutive nodes on the grid, and they may extend to different resolutions of cell levels. Known areas of failure can easily be bypassed during data point selection using the transformation graph described in Section 3, by setting the appropriate edge capacities to infinity. For example, if cell 4 failed, setting the capacity of edge $(4, C)$ to ∞, would force the algorithm to select a different solution.

Note that areas can fail in such a way that the failures cannot be bypassed for some queries. For the query of Figure 3a, if areas 2 and 4 fail, there is no way of retrieving the exact answer. Note that in this case there would be no cut in the flow graph with weight $< \infty$. It is interesting to note that it is possible to compute the total aggregate value in the combined $\{2, 4\}$ area, but the query only intersects part of that region.

Using the flow reduction, it can be therefore determined in polynomial time whether the query can be answered accurately or not. Figure 6a shows an example where the query value can be computed accurately. In the cases where computing the accurate answer is infeasible, a solution should be found that best approximates it. Without further knowledge of correlations or other information on the actual data, a reasonable approach is to assume uniformity over the values in the failed region and use it to estimate the aggregate



(a) (b)

Fig. 6. (a) Example query that can be computed accurately (b) Example query that cannot be computed accurately

in the portion of it that is requested by the query. To minimize inaccuracies due to the uniformity assumption, this estimation should be done in as small a portion of the failed region as possible, relative to the region of interest. An interesting issue for future work is to further investigate various approaches to construct estimates for non-recoverable regions.

In the example of Figure 6b the query intersects only $\frac{1}{3}$ of the failed region, and the exact answer cannot be retrieved. However, instead of estimating the sum of the overlapping region as $\frac{1}{3}$ of the full failed area, it is probably better to estimate it as $\frac{1}{2}$ of the top 2-cell portion of the failed region, which can be accurately reconstructed.

Algorithm 4 traverses a cube hierarchy like the ones in Figure 6 bottom up, starting with an area A matching the query region that contains failures. If the sum over A cannot be computed, the algorithm traverses to higher levels and the area A gets augmented as necessary. The algorithm returns the exact answer if eventually A does not get augmented, otherwise returns an estimate based on the uniformity assumption, over the portion of the failed region that has been recovered.

Algorithm 4 Region Recovery

Start at leaf level failed regions Q_A. Initialize $V = 0$ and $A = Q_A$;
while fail(currentNode) **do**
  level$--$;
  if $fail(children(currentNode)) > 1$ **then**
    $A = A \cup newLeafFailures$;
    $V = V - \sum V(aliveLeavesInNewFailure)$;
  end if
  $V = V - \sum V(aliveChildren)$;
  if !$fail(currentNode)$ **then**
    $V = V + V(currentNode)$;
  end if
end while
if $A > Q_A$ **then**
  region Q_A cannot be fully recovered
end if
return $estimate = V\frac{Q_A}{A}$

6 Related Work

Spatial query processing has been extensively studied in centralized systems. The R-tree [10], and its variants ([11, 12]), is one family of index structures for spatial data. In the R-tree, each spatial data object is represented by a Minimum Bounding Rectangle containing a pointer to the object in the database. Non-leaf nodes store a MBR that contains the MBRs of all the children nodes. A query traverses the R-tree using "containment"

and "overlap" checks to appropriately navigate through the structure. Spatial indexing is discussed more extensively in [13].

Because of resource limitations in sensor networks, building centralized indexes is often not practical. [14] proposes a peer-tree, a distributed R-tree using peer-to-peer techniques, partitioning the sensor network into hierarchical, rectangle shaped clusters. The techniques include joins and splits of clusters, and the authors show how to use the structure to answer nearest neighbor queries. The use of quad-trees in such setting is also natural and [15] uses distributed quad-trees to support spatial querying.

SPIX [16], is a distributed spatial index which uses Minimum Bounding Areas in an R-tree like fashion. The spatial query processor on each sensor uses SPIX to bound the branches that do not lead to results, find a path to sensors that do have results to report, as well as aggregate data.

In the database community, a number of distributed and push-down based approaches have been proposed for aggregation ([17, 18]). However these assume a well connected, low-loss topology that is often unrealistic in sensor networks. TAG ([19, 20]) performs in-network aggregation to reduce the amount of data that needs to be send over the network. TAG provides a simple declarative interface for aggregation, and it distributes and executes aggregation operators in the network, computing aggregates through data flow. Aggregation in adversarial settings is examined in [21].

We use ideas from Data Cubes [6] and simple summarization techniques [9]. The aR-trees [22] introduce OLAP type aggregation indexes over spatial data, resembling our data cube hierarchies, but are not oriented to a distributive environment, making some construction and failure issues irrelevant. Our approach can also relate to methods of distributed storage in sensor networks [23, 24], where data get disseminated and encoded at different locations to improve resilience to failures. Our approach does not simply target failures, but also improves processing for spatial aggregates, which commonly occur in these settings.

7 Conclusions and Future Directions

In this work we presented multi-resolution cube estimators that keep summary information distributed in the network, and facilitate the computation of spatial aggregate queries. We presented algorithms that compute query plans over the cube hierarchies in polynomial time, and showed how they can also accommodate multiple queries. We presented a distributed scalable algorithm that performs the cube construction and algorithms that perform failure recovery. We demonstrated that multi-resolution cubes are very resilient to node failures, and also behave well in larger region outages.

In this paper we focused on a grid topology, but it is an interesting problem to extend this approach as an overlay over any deployment. Using observations at the actual sensor locations and spatial models of the data distributions, we can infer the values at the grid locations. This would require us to adapt our algorithms to account for probabilistic data and the possible correlations across the grid locations. Especially in the case of prefix-sum cubes, which maintain summaries of overlapping areas, these correlations cannot be disregarded. Simple sum cubes are easier to extend as the summary regions do not intersect, making the problem less complex.

References

1. Madden, S., Gehrke, J.: Query processing in sensor networks. Pervasive Computing **3**(1) (2004)
2. Deshpande, A., Guestrin, C., Madden, S., Hellerstein, J., Hong, W.: Model-driven data acquisition in sensor networks. In: VLDB. (2004)
3. Meliou, A., Chu, D., Guestrin, C., Hellerstein, J., Hong, W.: Data gathering tours in sensor networks. In: IPSN. (2006)
4. Meliou, A., Krause, A., Guestrin, C., Hellerstein, J.M.: Nonmyopic informative path planning in spatio-temporal models. In: AAAI. (2007)
5. Meliou, A., Guestrin, C., Hellerstein, J.M.: Approximating sensor network queries using in-network summaries. In: Information Processing in Sensor Networks (IPSN). (2009)
6. Gray, J., Chaudhuri, S., Bosworth, A., Layman, A., Reichart, D., Venkatrao, M., Pellow, F., Pirahesh, H.: Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. J. Data Mining and Knowledge Discovery **1**(1) (1997) 29–53
7. Dimakis, A.G., Prabhakaran, V., Ramchandran, K.: Ubiquitous access to distributed data in large-scale sensor networks through decentralized erasure codes. In: IPSN, Piscataway, NJ, USA, IEEE Press (2005)
8. Xu, K., Takahara, G., Hassanein, H.: On the robustness of grid-based deployment in wireless sensor networks. In: IWCMC. (2006) 1183–1188
9. Hillis, W.D., Steele, Jr., G.L.: Data parallel algorithms. Commun. ACM **29**(12) (1986) 1170–1183
10. Guttman, A.: R-trees: a dynamic index structure for spatial searching. In: SIGMOD, ACM (1984) 47–57
11. Beckmann, N., Kriegel, H.P., Schneider, R., Seeger, B.: The R*-tree: an efficient and robust access method for points and rectangles. In: SIGMOD, ACM (1990) 322–331
12. Sellis, T.K., Roussopoulos, N., Faloutsos, C.: The R+-tree: A dynamic index for multi-dimensional objects. In: VLDB. (1987) 507–518
13. Güting, R.H.: An introduction to spatial database systems. The VLDB Journal **3**(4) (1994) 357–399
14. Demirbas, M., Ferhatosmanoglu, H.: Peer-to-peer spatial queries in sensor networks. In: P2P, IEEE Computer Society (2003) 32
15. Demirbas, M., Lu, X.: Distributed quad-tree for spatial querying in wireless sensor networks. ICC (2007)
16. Soheili, A., Kalogeraki, V., Gunopulos, D.: Spatial queries in sensor networks. In: GIS, ACM (2005) 61–70
17. Shatdal, A., Naughton, J.F.: Adaptive parallel aggregation algorithms. SIGMOD Rec. **24**(2) (1995) 104–114
18. Yan, W.P., Larson, P.A.: Eager aggregation and lazy aggregation. In: VLDB. (1995) 345–357
19. Madden, S., Franklin, M.J., Hellerstein, J.M., Hong, W.: TAG: a tiny aggregation service for ad-hoc sensor networks. SIGOPS Oper. Syst. Rev. **36**(SI) (2002) 131–146
20. Madden, S., Szewczyk, R., Franklin, M.J., Culler, D.: Supporting aggregate queries over ad-hoc wireless sensor networks. In: WMCSA. (2002) 49
21. Garofalakis, M.N., Hellerstein, J.M., Maniatis, P.: Proof sketches: Verifiable in-network aggregation. In: ICDE, IEEE (2007) 996–1005
22. Papadias, D., Kalnis, P., Zhang, J., Tao, Y.: Efficient OLAP operations in spatial data warehouses. Advances in Spatial and Temporal Databases (2001) 443–459
23. Aly, S.A., Kong, Z., Soljanin, E.: Fountain codes based distributed storage algorithms for large-scale wireless sensor networks. In: IPSN. (2008) 171–182
24. Dimakis, A.G., Prabhakaran, V., Ramchandran, K.: Decentralized erasure codes for distributed networked storage. IEEE/ACM Trans. Netw. **14**(SI) (2006) 2809–2816