Similarity-Aware Query Allocation in Sensor Networks with Multiple Base Stations

Shili Xiang[1], Hock-Beng Lim[2], Kian-Lee Tan[1], Yongluan Zhou[3]
[1]Department of Computer Science, National University of Singapore
[2]Intelligent Systems Center, Nanyang Technological University
[3]School of Computer and Communication Sciences, EPFL, Switzerland

ABSTRACT

In this paper, we consider a large scale sensor network comprising multiple, say K, base stations and a large number of wireless sensors. Such an infrastructure is expected to be more energy efficient and scale well with the size of the sensor nodes. To support a large number of queries, we examine the problem of allocating queries across the base stations to minimize the total data communication cost among the sensors. In particular, we examine similarity-aware techniques that exploit the similarities among queries when allocating queries, so that queries that require data from a common set of sensor nodes are allocated to the same base stations. We first approximate the problem of allocating queries to K base stations as a max-K-cut problem, and adapts an existing solution to our context. However, the scheme only works in a static context, where all queries are known in advance. In order to operate in a dynamic environment with frequent query arrivals and termination, we further propose a novel similarity-aware strategy that allocates queries to base stations one at a time. We also propose several heuristics to order a batch of queries for incremental allocation. We conducted experiments to evaluate our proposed schemes, and our results show that our similarity-aware query allocation schemes can effectively exploit the sharing among queries to greatly reduce the communication cost.

1. INTRODUCTION

Sensor nodes are small, inexpensive, low power and programmable. As such, there has been increasing adoption of wireless sensor networks (WSNs) in a wide variety of applications. WSNs have been used in resource-limited and harsh environments, such as earthquake areas, ecological contamination sites, and military battlegrounds. They have also been deployed in everyday-life environments, such as smart home environment, intelligent museum/zoo, warehouse/port, and road. WSNs are expected to greatly improve our understanding of the world and also provide our life with tremendous convenience.

To ease the deployment of WSN applications, researchers have proposed techniques to treat the sensor network as a database. In a database context, a typical WSN operates as follows: user interests are expressed as queries and submitted to a powerful base station; the base station disseminates the queries into the sensor network, more specifically, to the sensor nodes that are involved in the queries; the sensor nodes generate, process and transfer sensory data back to the base station which will then correspondingly return the query result back to the users. However, sensor nodes are quite resource-constrained with limited processing capacity, storage, bandwidth and power. To better realize the potential of WSNs, several query processing techniques have been specially designed to optimize the processing of each query [18, 12, 13, 4]. For example, for aggregation queries, a tree-like routing structure is often utilized [19, 9] and in-network aggregation is done at intermediate nodes to reduce the amount of data that is sent to the base station.

As the popularity and importance of WSNs grow, so does the size of the sensor network and the number of users who are interested in accessing sensory data. To further enable the sensor network to scale well with the number of users and queries, multiple query optimization techniques have also been investigated [16, 15], where they effectively share the commonality among the queries that are distributed to a particular base station to minimize the energy consumption among sensors. For a large scale sensor network, it is necessary and beneficial to have multiple base stations in the network. Firstly, it provides the sensor network with better coverage and scalability. The limited radio range of sensor nodes leads to multi-hop routing, where the nodes nearer to the base station need to do more work. With limited bandwidth of each sensor node, the nodes near the base station form the bottleneck and correspondingly the maximum number of hops each base station can cover is limited. Secondly, it provides the sensor network with better reliability [10]. The communication among sensor nodes are prone to failures, due to collision, node failure and environmental noise etc. With more base stations in the network, the average number of hops each data travels is fewer, and correspondingly the reliability of the data transmission is better. Lastly, it extends the life time of the sensor network. The sensor nodes nearer to the base stations are likely to have higher load and the energy consumption there is greater than other nodes; with more base stations, the burden of nodes nearer to each base station can be relieved.

In this paper, we adopt an infrastructure with multiple base stations to support large scale sensor network applications, in terms of both network size and number of user queries. We are the first to formulate and deal with the query allocation problem in such a context, that studies how queries should be allocated onto various base stations to minimize the total communication cost among sensor nodes. More specifically, during the query allocation process, our al-

Proceedings of the 4th International Workshop on Data Management for Sensor Networks (DMSN'07), Vienna, Austria, 2007.

gorithms are aware of the similarities among the queries. We approximate the query allocation problem as a Max-K-Cut problem, and adapt a classical solution for Max-K-Cut that uses Semidefinite Programming (SDP) relaxation to solve it [6]. However, the SDP approach is designed for static environment where all queries are known apriori. In our context, new queries may arrive and running queries may terminate. This calls for novel techniques to be designed. In addition, as we shall see, the Max-K-Cut does not exactly capture our query allocation problem. Therefore, we propose a novel incremental query insertion algorithm that incurs little overhead to allocate a newly inserted query to an appropriate base station. For the situation where several queries arrive at the same time (or several queries are in the waiting queue), such as the initial setup, we propose several techniques to order these queries before inserting them into the network one by one.

Our experimental results show significant improvement of all our similarity-aware query allocation schemes, over the best allocation strategy that fails to take advantage of the inherent sharing among other queries. Moreover, our proposed incremental query insertion algorithms perform as well as the complex Max-K-Cut classical solution in terms of the communication cost among sensor nodes, while incurring negligible computational time.

The rest of this paper is organized as follows. In Section 2, we formulate our query allocation problem and point out the challenges to solve the problem. Then, we approximate the query allocation problem with a Max-K-Cut problem in Section 3. To deal with dynamic query insertion and termination, in Section 4, our incremental insertion algorithms are proposed and discussed. In Section 5, we present our experimental results. Before we finally conclude the paper in Section 7, we review some related work in Section 6.

2. PROBLEM FORMULATION

Consider a large scale sensor network that comprises K base stations and hundreds of (say N) sensor nodes. The base stations are powerful machines such as a desktop/server, with abundant processing, storage, and memory capacity and can be recharged easily. On the other hand, the sensor nodes are resource constrained, with limited processing capacity, storage, bandwidth and power. Thus, it is vital to sensor network applications to conserve the resources of sensor nodes. In this work, we focus on the communication cost among sensor nodes. Since the base stations are assumed to be wired, we do not consider the communication cost among them. Moreover, we assume that queries are submitted at the wired part of the network, and are allocated to a base station for processing.

For a set of queries that are allocated to a base station, we assume the existence of a multiple query optimization scheme that can effectively exploit the common requests and share them among various queries. There are existing works on multiple query optimization in sensor networks, such as [16, 15]. In this paper, we exploit the inherent sharing among queries when we allocate them to the base stations. To integrate our query allocation work with a specific multiple query optimization scheme that runs at each base station, we just need to incorporate its cost model which reflects how much sharing that particular scheme can achieve.

Our query allocation problem is defined as follows. Suppose there are K base stations and currently M queries are running in the sensor network. For a query q_i running exclusively at a specific base station b_j, a number of radio messages will be transmitted to retrieve the sensory data to the base station. We refer to the number of radio messages as the communication cost. Let c_{ij} denote the communication cost incurred by a query q_i at base station b_j. To estimate the cost, we need the distances from the queried sensor nodes to the base station, as well as the probability that each queried node has data that satisfy the query. The estimation of such probability can be achieved by maintaining statistical models at the base station, which is an independent problem that has been studied in other literatures, such as [5]. To simplify the probability estimation, in this paper, we use region-based aggregation queries. We further denote the query set allocated to base station b_j as Q_j, and the amount of sharing (redundant requests) among these queries as S_j. Then the objective of the query allocation problem is to minimize the communication cost among sensor nodes in order to answer the queries. More formally, the objective function can be expressed as:

$$minimize \quad \sum_{j=1}^{K}\Big(\sum_{q_j \in Q_j} c_{ij} - S_j\Big)$$

To facilitate the in-network aggregation, in our paper, each base station is the root of its respective routing infrastructure, e.g., a routing tree or routing Directed Acyclic Graph (DAG). The nodes in each routing infrastructure are the sensor nodes that are involved in the queries that have been allocated to the respective base station. Hence, to estimate the value of S_j, we identify the sum of exclusive sharing among queries in Q_j as follows. We keep bitmap m_j of size N maintained at base station b_j, whose default value is all zero. If a sensor node x is queried by q_i, where $q_i \in Q_j$, we check the value of $m_j[x]$. If it is 0, we set it to be 1. Otherwise, some other queries have already requested data from a sensor node x at base station b_j, and this cost is shared and correspondingly we add 1 to S_j.

If the multiple query optimization scheme at each base station does not exist, i.e., $S_j = 0, \forall j = 1...K$, the above optimal query allocation is easy to achieve in linear time. We could just allocate each query q_i to the base station b_j that incurs the least c_{ij}. That is,

$$q_i \in Q_j, \quad if \ c_{ij} = min(c_{i1}, c_{i2}, ..., c_{iK})$$

However, since multiple query optimization will be used at each base station, the introduction of S_j makes the query allocation NP hard. To get the optimal allocation, we need to get the optimal balance between minimizing $\sum_{j=1}^{K}\sum_{q_j \in Q_j} c_{ij}$ and maximizing $\sum_{j=1}^{K} S_j$.

Figure 1 shows an example of how the query cost c and sharing S are computed in our model. Each of the small circles denotes one sensor, while each rectangular region represents one query and each triangle denotes a base station. For an aggregation query q_i assigned to a particular base station b_j, c_{ij} is computed as the sum of the area size of the query region and the extra cost incurred by relaying the aggregated result back to the base station. For example, as illustrated in Figure 1, q_1 covers 25 sensors and its minimal distance to b_1 is 5, we denote c_{11} as 30. Similarly, $c_{12} = 28$. If both q_1 and q_3 are allocated to b_1, the regions E_5 and E_7 can be shared, hence $S_1 = E_5 + E_7$. It is worth noting that

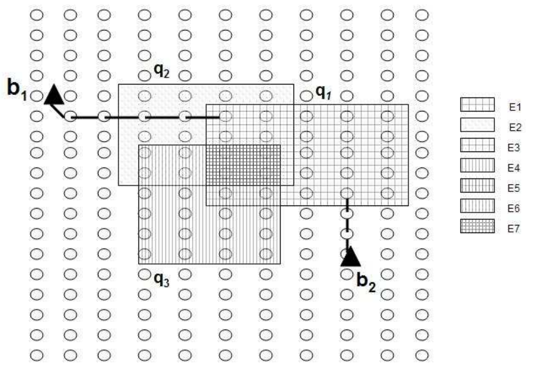


Figure 1: A Scenario with Multiple Base Stations and Queries

when q_1, q_2 and q_3 are all allocated to b_1, since E_7 has been shared twice, $S_1 = E_3 + E_5 + E_6 + 2 * E_7$.

3. MAX-K-CUT APPROXIMATION

From the above discussion, it can be seen that, to minimize the communication cost among sensor nodes, the queries should be allocated to achieve the followings. First, similar queries should be assigned to the same base station so that overlapping data need not be transmitted multiple times. This can reduce the energy consumption. Second, each query should be assigned to the base station that incurs the least communication cost. In other words, we could restate the problem as avoiding the following allocation as much as possible. First, similar queries are allocated onto different base stations. Second, a query is assigned to a base station that results in high communication cost.

Hence, in this section, we approximate the query allocation problem as a classical Max-K-Cut problem, with the objective to avoid these bad cases.

We construct a graph G = (V, E, W), where each vertex v_i represents either a base station or a query. To allocate a set of M queries to K base stations, we just need to partition V ($|V| = M + K$) into K subsets, where each base station belongs to only one partition. There is one edge e_{ij} between each pair of vertices v_i and v_j. The weight of an edge e_{ij} is given by w_{ij}, and $w_{ij} = w_{ij}$ (B denotes the set of base stations).

$$w_{ij} = \begin{cases} c_{ij} & \text{if} \quad v_i \in Q \quad and \quad v_j \in B; \\ -s_{ij} & \text{if} \quad v_i \in Q \quad and \quad v_j \in Q; \\ \infty & \text{if} \quad v_i \in B \quad and \quad v_j \in B. \end{cases}$$

As discussed earlier, c_{ij} denotes the cost of executing query q_i on base station b_j. s_{ij} denotes the amount of common requests between query q_i and q_j. The query allocation problem can then be expressed as a Max-K-Cut problem. That is, we partition V into K subsets. Formally, if we denote a partition P as P_1, P_2,...P_K, the problem is:

$$maximize \quad w(p) = \sum_{1<=l<r<=K} \sum_{i \in P_l, j \in P_r} w_{ij}$$

The rationale behind the formulation is as follows: the higher the value of c_{ij}, the higher the probability of edge e_{ij} being in the cut, and hence it will be less likely for q_i to be assigned to b_j. We use ∞ to denote the weight between each pair of base stations, so that different base stations are cut into different partitions. We use $-s_{ij}$ instead of s_{ij} to denote the weight of edge of e_{ij}, so that the more similar two queries are, the less likely the two queries will be separated into different partitions and allocated to different base stations. Moreover, if both q_i and q_j are assigned to the same base station b_k, the cost of the common data s_{ij} is counted both in w_{ik} and w_{jk}, with w_{ij} being $-s_{ij}$, it can eliminate the over-counted cost because multiple query optimization algorithm at b_k can enable the sharing and the same piece of data will only be transmitted once in the actual sensor network.

3.1 SDP K-cut

Max Cut is a well-known NP hard problem, and Max-K-cut is even more complicated than the Max Cut problem. According to [2], there can be no polynomial-time approximation scheme for Max-K-cut, for any $k >= 2$, unless P = NP. Goemans and Williamson [7] significantly improved the approximation rate from 0.5 to 0.878 for Max Cut problem by using SemiDefinite Programming (SDP) as a relaxation. In [6], Frieze and Jerrum extended the work to solve the Max-K-Cut problem, and achieves the expected approximation rate of α_K, where $\alpha_K - (1 - K^{-1}) \sim 2K^{-2} lnK$. We apply the algorithm in [6] to allocate a set of static queries onto their respective base stations, and denote the algorithm as SDP-K-Cut.

SDP-K-Cut is in fact a randomized heuristic algorithm using semidefinite programming relaxation that produces a K-partition which is provably better on average than the one produced by oblivious random partition. The challenge lies in how to model the variables which take one of K values. This is done by allowing y_i to be one of K vectors $a_1, a_2, ..., a_K$ defined as follows: take an equilateral simplex Σ_K in R^{K-1} with vertices $b_1, b_2, ..., b_K$. Let $c_K = \frac{\sum_{i=1}^{K} b_i}{K}$ be the centroid of Σ_K and let $a_i = b_i - c_K$ for $1 \le i \le K$, with scaled Σ_k so that $|a_i| = 1$ for $1 \le i \le K$. By proving that $a_i \cdot a_j = -1/(K-1)$ for $i \neq j$, the Max-K-Cut problem can be formulated as follows:

$$\begin{array}{lll} IP_K : & maximize & \frac{K-1}{K} \sum_{i<j} w_{ij}(1 - y_i \cdot y_j) \\ & subject \ \ to & y_j \in \{a_1, a_2, ..., a_K\}, \quad \forall j. \end{array}$$

By replacing y_i by v_i, where v_i can now be any vector in S_{n-1} so that there are more freedom to partition the space, they finally relax the max-K-cut problem into a semidefinite programming problem:

$$\begin{array}{lll} SDP_K : & maximize & \frac{K-1}{K} \sum_{i<j} w_{ij}(1 - v_i \cdot v_j) \\ & subject \ \ to & v_j \in S_{n-1}, \quad \forall j \\ & & v_i \cdot v_j \ge \frac{-1}{K-1}, \quad \forall i \neq j. \end{array}$$

The K partitions can now be obtained after the following two steps:

1. Solve the problem SDP_k to obtain vectors $v_1, v_2, ..., v_n \in S_{n-1}$
2. Choose K random vectors $z_1, z_2, ..., z_K$. Partition V into $P_1, P_2, ..., P_K$ according to which of $z_1, z_2, ..., z_k$ is closest to each v_j. That is,

$$P_i = \{j : v_j \cdot z_i \ge v_j \cdot z_{i'}, for \ all \ i' \neq i\}, \ for \ 1 \le i \le K$$

3.2 Implementation of SDP-K-Cut

To implement the SDP-K-Cut algorithm, the challenge lies in solving the problem SDP_K (Step 1). We adopt the convex programming form of SDP_K and use the SDPT3 [14], a solver for semidefinite-quadratic-linear programming developed by Toh et. al to obtain the result vectors.

More specifically, we denote $x_{ij} = v_i \cdot v_j$, and hence the SDP_K problem can be represented as:

$$\begin{array}{lll} CP_K: & minimize & \sum_{i<j} w_{ij} * x_{ij} \\ & subject\ to & x_{jj} = 1, \ \forall j \\ & & x_{ij} \geq \frac{-1}{K-1}, \ \forall i \neq j. \end{array}$$

Due to the constraint $x_{ij} \geq \frac{-1}{K-1}$ for $i \neq j$ instead of normal constraint $x_{ij} > 0$, we need to solve another linear programming problem together with the semidefinite programming problem. Since X is symmetric, we set $|V| * |V-1|/2$ linear constraints to especially deal with the situation for off-diagonal. Then, we can use SDPT3 to solve this problem. After we get result x_{ij}, since X is a symmetric positive semidefinite matrix, we can get vectors v by Cholesky factorization.

4. INCREMENTAL ALGORITHMS

The above Max-K-Cut approximation is not ideal. Firstly, by approximating our query allocation problem as a Max-K-Cut problem, the general trend of the query allocation is captured. However, the Max-K-Cut model does not adequately reflect all the properties of our problem. With the sum of the edge weights of the whole network fixed, the sum of the edge weights in each partition will be minimized when we get the maximum K cut. However, the edge weights in our problem are not entirely independent - they may be related to other edge weights and they may have overlap. For example, if some set of data is shared by three queries, and these three queries are all allocated to the same base station, by adding together all the weights of all the edges in the partition, the cost of the set of data will be deducted by three times while actually only two times should be deducted. Secondly, in a dynamic context, new queries continue to arrive while currently running queries may terminate at any time. The SDP K-cut solution, which is designed to solve the max K-cut problem for a static graph, is not an incremental algorithm. Upon every insertion of a query, the SDP K-cut algorithm will compute from scratch instead of incrementally optimizing the new query set based on the previous status. Hence, it is computationally impractical to deploy the SDP K-cut in a dynamic context.

Hence, in this section, we will introduce our own heuristics for the query allocation problem. Our solution is incremental, and aims at efficiently allocating queries upon their insertion while minimizing the communication cost among sensor nodes in the whole network.

4.1 Insertion Algorithms

Consider a newly arrived query q_i that needs to be inserted into the sensor network. When it arrives at base station b_x, b_x works as the coordinator. The coordinator b_x performs the query allocation as follows:

1. Estimate the additional cost ac_{ij} of executing q_i at b_j. This can be easily achieved with the bitmap m_j maintained by each base station as we mentioned in Section 2. For the region-based aggregation queries studied in this paper, the coordinator just need to obtain the bitmaps from q_i's neighbor base stations b_js.
2. Put q_i to the b_j that results in the smallest a_{ij}.

Here we use the additional cost ac_{ij} as the metric instead of the amount of sharing q_i that can be benefited from other queries at the base station b_j. This is because the additional cost is more effective in reducing the total cost: it not only reflects the amount of non-sharing region, but also reflects the cost of executing q_i at b_j by itself.

Now, it is possible for several queries to be available at the waiting queue (e.g., several queries arrive at the system at the same time or the system decides to batch queries for allocation). In this case, we still insert queries one at a time. However, we also study the following three heuristics to determine if the ordering of insertion may affect performance.

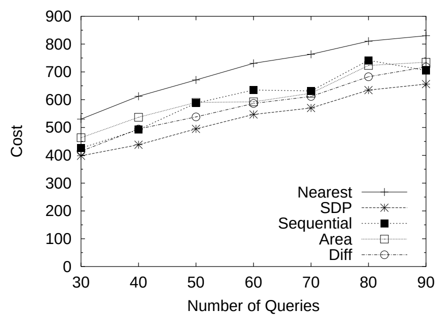
- **Sequential**: we insert these queries based on their submitted sequences.
- **Diff**: we order the batch of queries before they are inserted in the sensor network, in the descending order of the minimal difference of allocating this particular query onto different base stations. That is, for each query q_i, if $c_{ij} == MIN(c_{i1}, c_{i2}...c_{iK})$, and $c_{im} == MIN(c_{i1}, ...c_{ij-1}, c_{ij+1}, ...c_{iK})$, the minimal difference of query q_i is denoted as $c_{im} - c_{ij}$. The intuition behind this heuristic is as follows: if a query incurs more overhead than other queries upon its sub-optimal allocation, it implies that it is much nearer (in terms of cost) to one of the base stations, and hence we hope to allocate it first to the base station nearer to it before examining other queries.
- **Area**: we order the batch of queries before they are inserted in the sensor network, in the descending order of the area of their query region. That is, to insert "BIG" queries first. In this way, we suppose the queries with bigger regions will have the opportunity to be allocated to their nearest base stations; smaller queries that are inserted later are also likely to benefit from such "big queries" since the probability of finding an existing query that shares overlapping regions becomes higher.

As we discussed in Section 2, if we are not aware of the similarity among other queries that have been allocated to a base station, the best possible allocation is: for each query q_i, put it onto the base station b_j that incurs the smallest cost, i.e., $w_{ij} == MIN(w_{i1}, w_{i2}...w_{iK})$. We denote this allocation as **Nearest**. Note that even though **Nearest** is oblivious of the query similarity during allocation, queries allocated to the same base station may still benefit from data sharing through the multiple query optimization at the base station.

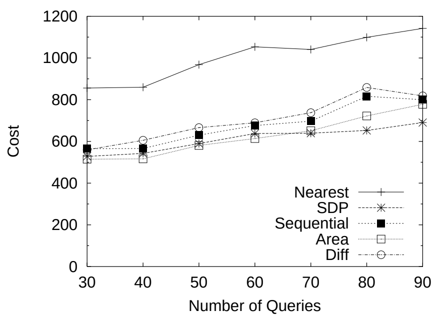
5. EXPERIMENTAL STUDY

In this section, we shall present experimental results to show the performance of our schemes. We evaluate the algorithms by varying the number of base stations and queries, and the average size of query regions.

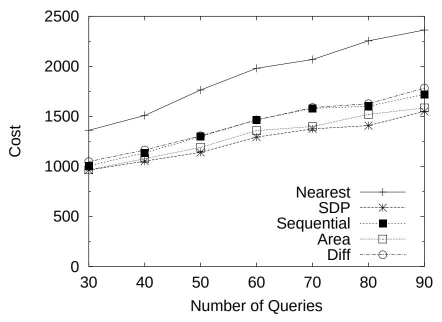
In the experiments, we assume N sensor nodes are deployed uniformly in a two-dimensional grid square. For every 100 sensor nodes, there is one base station at the center. Each query is expressed as a rectangular region $((x_1, x_2), (y_1, y_2))$, where (x_1, y_1) is randomly selected from



(a)N=400, QR=5*5



(b)N=400, QR=8*8



(c)N=900,QR=8*8

Figure 2: Communication Cost over Random Queries with varying network size and average region

any point in the network, and the lengths on the x-axis ($x_2 - x_1$) and y-axis ($y_2 - y_1$) satisfy the uniform distribution or gaussian distribution. We assume lossless communications among the sensor nodes, to evaluate the actual gain brought by our similarity-aware query allocation algorithms. We simulate the behaviors of base stations and sensor nodes on a 3 GHz Intel Pentium IV machine with 1 GB of main memory running Windows XP.

To compare the effectiveness of our incremental algorithm against SDP-K-cut (which is designed for static environment where all queries are known apriori before being allocated), we evaluate the schemes under a static environment here. In other words, for our schemes, we assume a batch of queries is available, and they are inserted into the network one at a time (based on the heuristics used). Each of the results shown below is the average result after 10 runs.

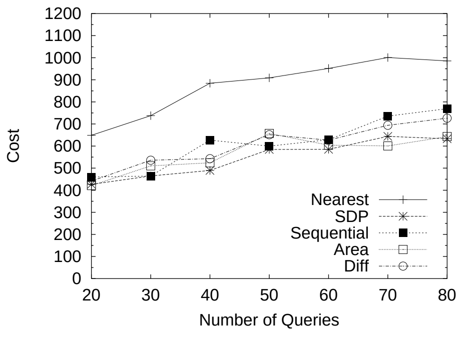


Figure 3: Communication Cost over Queries with axis length∼N(8,5)

Figure 2 shows the results when the lengths of query regions are generated uniformly, while Figure 3 shows the results when the lengths of query regions are generated using the Gaussian distribution. From the experimental results in Figures 2 and 3, we observe that the **nearest** strategy performs the worst. This is expected since it does not take advantage of the sharings that can be exploited among queries. On the other hand, all the similarity-aware schemes (max-K-cut and our similarity-aware algorithms) result in a significant reduction in communication cost. We also observe that our proposed variants perform nearly as well as the complicated max-K-cut classical solution. As for the three similarity-aware variants, although **area** performs better in most cases, none of them shows definite advantages over the others and the differences are not huge. This suggests that the key performance reduction comes from the inherent sharing and similarity among queries, and the ordering of queries in a batch offers only marginal benefit, if any.

Comparing Figure 2(a) with Figure 2(b), we note that with the same number of queries, queries with larger average regions are likely to benefit more. This is because there are more overlaps among queries when each query is querying a larger region, and hence more benefit can be exploited by our similarity-aware query allocation algorithms. Using the same logic, referring to Figure 2(b) and Figure 2(c), with the same number of queries and query regions with the same average size, more gain is achieved when the network size is smaller.

Recall that the lengths of query regions in Figures 2(b) and 3 are generated using different distributions. In both cases, the average query region is 64. Looking at the results, we observe that the relative performance among the various schemes are the same in both cases. Thus, our query allocation algorithms can effectively capture the sharings even under different distributions for query lengths.

Figure 4 shows the computational time taken to solve the Semidefinite program in the SDP K-cut solution. The computational time increases exponentially when the number of queries increases. As the number of queries reaches 90, the time to get the partition is more than 10 minutes, in a network with 900 sensors (9 base stations). For our similarity-aware variants, it takes less than 4ms to compute the query allocation. This overhead is negligible, making our proposed methods attractive for dynamic environments.

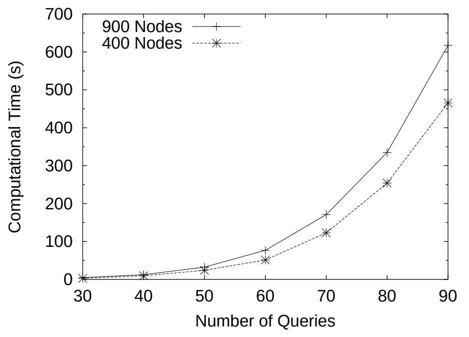


Figure 4: CPU Time for SDP_K over Random Queries with Average QR=8*8

Before leaving this section, it is worth noting that no matter whether the query allocation algorithm is similarity-aware or not, the queries that are allocated to the same base station exploited sharing in processing the queries. As such, together with the savings from in-network aggregation, even the **nearest** algorithm largely outperforms the naive scheme where each sensor node sends its raw data to its nearest base station (for the base station to process the queries). We did not present these results here.

6. RELATED WORKS

Query/operator allocation has been studied in both traditional distributed database systems [8] and, more recently, stream processing systems [1, 17, 20]. These optimization algorithms mainly focused on fine tuning the allocation of queries/operators across the distributed servers to balance the load distribution as well as to minimize the communication cost among the servers. However, the context of this paper is much different from theirs. Our objective is to minimize the communication cost inside the sensor network instead of among the base stations. Moreover, we endeavor to maximize the sharing of the data collection cost among various queries allocated to the same base station. This is typically not considered in existing work.

This paper is also related to some other algorithmic literatures. Note that the query allocation problem can be solved by a two-phase approach: a query partitioning phase followed by a mapping phase. In the query partitioning phase, queries are partitioned into K disjoint sets, so that the amount of sharing in the K partitions is maximized. In the mapping phase, it is in fact a complete bipartite mapping problem [3], where K partitions are allocated to K base stations so that the weight of the mapping is minimized. Again, most of the existing work in query partitioning either balance the partitions or minimize the number or weight of cuts [11]. The latter category suffers from the similar problem as our Max-K-Cut approximation that we have discussed in Section 4. Hence, they are not ideal for our case.

7. CONCLUSION

In this paper, we have examined the query allocation problem in sensor networks with multiple base stations, and proposed several similarity-aware algorithms to minimize the total data communication cost among the sensor nodes. Experimental results show that our similarity-aware query allocation schemes can effectively exploit the sharing among queries and greatly reduce the communication cost. However, with our current incremental insertion algorithms, when a query is inserted, we just do the best for the newly inserted query, but do not allow any migration of running queries that already existed in the system. This means that the allocation may not be globally optimal. To deal with this problem and also to deal with the effect of termination of queries that often happen in real systems, existing queries may need to be re-allocated if necessary. Thus, we plan to explore migration algorithms as part of our directions for future study.

8. REFERENCES

[1] Y. Ahmad and U. Çetintemel. Networked query processing for distributed stream-based applications. In *VLDB*, pages 456–467, 2004.

[2] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM (JACM)*, 45(3):501–555, 1998.

[3] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 2001.

[4] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos. Hierarchical in-network data aggregation with quality guarantees. In *EDBT*, 2004.

[5] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, 2004.

[6] A. Frieze. Improved approximation algorithms for max k-cut and max bisection. *Algorithmica*, 18(1):67–81, 1997.

[7] M. Goemans and D. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995.

[8] H. Lu and K. Tan. Load-balanced join processing in shared-nothing systems. *Journal of Parallel and Distributed Computing*, 23(3):382–398, 1994.

[9] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TinyDB: An acquisitional query processing system for sensor networks. *ACM TODS*, 30(1), November 2005.

[10] A. Munteanu, J. Beaver, A. Labrinidis, and P. K. Chrysanthis. Multiple query routing trees in sensor networks. In *Proc. of the IASTED International Conference on Databases and Applications (DBA)*, 2005.

[11] N. Selvakkumaran and G. Karypis. Multiobjective Hypergraph-Partitioning Algorithms for Cut and Maximum Subdomain-Degree Minimization. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 25(3):504–517, 2006.

[12] A. Silberstein and J. Yang. Many-to-many aggregation for sensor networks. In *ICDE*, 2007.

[13] X. Tang and J. Xu. Extending network lifetime for precision-constrained data aggregation in wireless sensor networks. In *INFOCOM*, 2006.

[14] K.-C. Toh, et al. Sdpt3. `http://www.math.nus.edu.sg/~mattohkc/sdpt3.html`.

[15] N. Trigoni, et al.Multi-query optimization for sensor networks. In *DCOSS*, 2005.

[16] S. Xiang, et al. Two-tier multiple query optimization for sensor networks. In *ICDCS*, 2007.

[17] Y. Xing, S. B. Zdonik, and J.-H. Hwang. Dynamic load distribution in the borealis stream processor. In *ICDE*, 2005.

[18] X. Yang, et al. In-network execution of monitoring queries in sensor networks. In *SIDMOD*, 2007.

[19] Y. Yao and J. Gehrke. Query processing for sensor networks. In *CIDR*, 2003.

[20] Y. Zhou, et al. Efficient dynamic operator placement in a locally distributed continuous query system. In *CoopIS*, 2006.