SEÇMECE: Optimizing Lifetime of Federated Sensor Networks by Exploiting Data and Model Redundancy

Alptekin Küpçü
Department of Computer Science
Brown University
Providence, RI 02912 USA
kupcu@cs.brown.edu

ABSTRACT

Next generation sensor network deployments are foreseen to be large infrastructures, with multiple concurrent tasks running on the same set of hardware. Applications will need standardized methods to access and integrate data from such heterogeneous sensor networks. Hence, a Federated Sensor Network (FSN) model can significantly simplify the development of multi-network applications by presenting a unified system view, hiding the heterogeneity, and performing optimizations.

The optimizations are possible due to redundancies in the system. First, a given task can be accomplished by multiple alternative ways (i.e. models), each of which incorporates different types of sources (model redundancy). Second, sensor readings about the same physical phenomenon at some well-defined spatial locations are correlated (spatial data redundancy), and third, sensor observations do not lose their utility immediately but only over time (temporal data redundancy).

Seçmece optimizer makes use of these redundancies to select which models to run, using which sources at which rate, aiming to achieve the required quality with the minimum cost. We first show that even this optimization problem is strongly NP-Hard, and then extend the problem to maximize the system lifetime, the time the system can continue answering the given queries with enough quality. Finally, we study our solution approaches, and possible modifications, to demonstrate performance gains in both real world scenarios and random generated setups.

1. INTRODUCTION

The term Sensor Network means any (generally wireless) network of (usually battery-powered) sensor devices (cameras, microphones, temperature sensors, blood pressure sensors etc.). Some of the applications developed under sensor networks research include security applications [17, 25, 9, 27, 20], medical applications [29, 23, 6, 7], and environmental monitoring applications [12, 2, 31]. These applications are often deployed for long-term usage.

Even though currently these applications are all considered separately and independently, in the near future, they can easily be combined to work together in a single large-scale ***Federated Sensor Network*** (FSN) deployment. The interaction with such a complex heterogeneous system should be unified and easy for the user. The system should take care of all the details, including efficiency issues.

Our goal is to design a framework capable of supporting FSN applications in a resource efficient and easy to use way. The optimization part of this framework, Seçmece optimizer, makes use of three types of redundancies. The first one is the ***model redundancy***, which relies on the observation that a given task can be accomplished in multiple possible ways (i.e. models) using different source types. Different models can offer different quality answers indicating the error (ex. temperature is $20^\circ C$ with $\pm 1^\circ C$ or $\pm 5\%$ error), or the belief value/probability associated with the answer (ex. the identity of the driver is 95% believed to be Mike), or resolution etc. depending on the context. Expectedly, there is a cost-quality trade-off.

Other types of redundancies in such a heterogeneous multi-sensor system are data redundancies. ***Spatial data redundancy*** relies on the observation that sensor readings at some well-defined spatial locations are correlated. For example, temperature readings of sensors close to each other are close. ***Temporal data redundancy*** relies on the usability of the readings over time. The reading can still offer a useful estimate over a period of time, where the length of the period and the quality decrease rate depends on the phenomenon and the model using that value.

In a system with many sources of many different types, and multiple queries waiting for answers simultaneously and continuously, with possible alternative models (i.e. ways) achieving different quality levels at variable costs, it becomes necessary and effective to optimize the system. The optimization process involves selecting which models to use for answering the queries, and for those models, which sources to use at which sampling rates (rates in short) to achieve enough quality with the least cost (defined on battery consumption). The goal of optimization in such a setting should be maximizing the system lifetime, which is not defined as the traditional network lifetime; the time until the first node dies or the first network partition occurs. ***System lifetime*** is the time the system can continue answering the queries with enough quality.

The contributions of this paper are giving motivating examples including security applications, medical applications,

and environmental monitoring using FSNs, defining the NP-Hard quality-based optimization problem, defining and analyzing the system lifetime as an optimization goal, designing heuristic solutions exploiting redundancies in the system, and studying and comparing the solution approaches.

2. FEDERATED SENSOR NETWORKS

2.1 Application Areas

We are interested in applications which incorporate many sensors of different types to answer multiple queries which can be answered in multiple alternative ways. The selection of what to use depends on the quality requirements that can be achieved at various costs. Using different quality requirements, other than just the highest possible quality, is useful since if some quality level is enough for the purpose, limiting it to that value would help improving the system lifetime. One might also want to achieve the highest possible quality during a given lifetime. This dual way of thinking is discussed in Section 5.2.

Security: One example of an FSN application is a generic security surveillance application. It can make use of cameras, motion sensors, microphones, light sensors, weight sensors, lasers, touch sensors etc. Different parts of the area may contain different sensors and may require different surveillance qualities. For example, a room without windows may not need a high surveillance quality. Cheaper sensors like lasers can be always-on to detect breakdowns, whereas cameras may be used only when a breakdown event is detected, assuming they give higher quality output at a higher cost. Furthermore, such a home security system can even be combined with an elderly care system [6, 7]. The health observation can require low quality for elderly health info when someone else is at home, higher otherwise. Similarly, we may need lower quality for security when people are at home and higher if all are sleeping or away.

Medical Applications: A personal-scale medical application [29] can use biological sensors like skin temperature sensors, actigraphs, ECG sensors, blood pressure sensors, blood sugar sensors, heart rate sensors, respiration sensors etc. for getting information about a person's health. The queries are monitoring the thermal, hydration, and cognitive status of the person, as well as the life signs. The thermal state can, for example, be obtained by a model using the skin temperature or another using the digested pill sensor to get the core body temperature. These different techniques offer different quality levels for varying costs. Besides, the quality requirements can change, depending on the current status of the person, where costly but high quality sensors should better be saved for critical situations. Furthermore, increased latency would dramatically decrease the information quality.

Thinking in larger scale, one can imagine a hospital deployment [23]. Patients will be equipped with bio-sensors and many doctors and nurses will be interested in patient information, continuously querying the system. Furthermore, there would be environmental sensors both for quality of life within the hospital (ex. air conditioning) and security purposes.

Environmental Monitoring: If we are interested in plants instead of humans, all of the above are also valid for precision agriculture applications [12, 2]. There are environment sensors, again for security and climate control, plus the bio-sensors and chemical sensors. We might be interested in either single plants or a group of plants of the same kind. Thus, information from different plants, which are obtained by using different types of sensor readings, might need to be combined, even with the environment information. Again, there may be different quality requirements depending on the plant types, and higher qualities may be required during crop flowering or fertilization times.

Now consider a volcano monitoring system [31]. Different sensor types like seismic activity sensors, special infrasonic microphones, temperature and light sensors, cameras and GPS can be incorporated. Less quality infrasonic microphones are less costly compared to higher quality seismic sensors, but multiple microphones can be used to improve detection quality. Latency is also an issue, since late detection will decrease the usefulness. Also when there are no signs of an activity, quality requirements are not too high, whereas after the detection of signs of an activity, one should use higher quality sensors to get better output.

All the applications above are just some examples where Seçmece optimization can be applied to improve the system lifetime. Seçmece is by no means the only solution to these applications, and for some of them further improvements might be achieved by using in-network computation or filtering mechanisms on top of Seçmece optimization.

2.2 System Definition

All the applications above share some properties that make them FSN applications: Heterogeneous devices are deployed and used in combination, tasks can be accomplished in multiple ways achieving quality requirements at various costs, multiple applications coexist with multiple users issuing multiple queries concurrently. An FSN framework should offer a user interaction layer and a network interaction layer to ease the interaction with users and with networks of different types of sources. The interaction with heterogeneous networks is handled by individual network managers for each sensor type, which deals with sensor specific issues like *turning* them *on/off*, setting their *rates*, forming *routing* trees, and collecting *sensor information* (location, battery, etc.). Details of those layers are not in the scope of this paper.

To present the optimization-related details of the framework, we will be using a parking lot sensor network deployment scenario [32] with our extensions. There are cameras, magnetometers, break beam sensors, smoke sensors, microphones, id card readers, etc. deployed. Sample usages of this multi-task system by different people are as follows:

- Police Officer Pat wants a photograph of all vehicles moving faster than 15mph.

- Employee Alex wants to know what time to arrive at work in order to get a parking space on the first floor of the parking deck.

- Safety Engineer Kim wants to know the speeds of cars near the elevator to determine whether or not to place a speed bump for pedestrian safety.

- Security Chief Matt wants to get a photo of the robber if any robbery happens at the parking deck, where such an event can also be detected by microphones, thanks to car alarms.

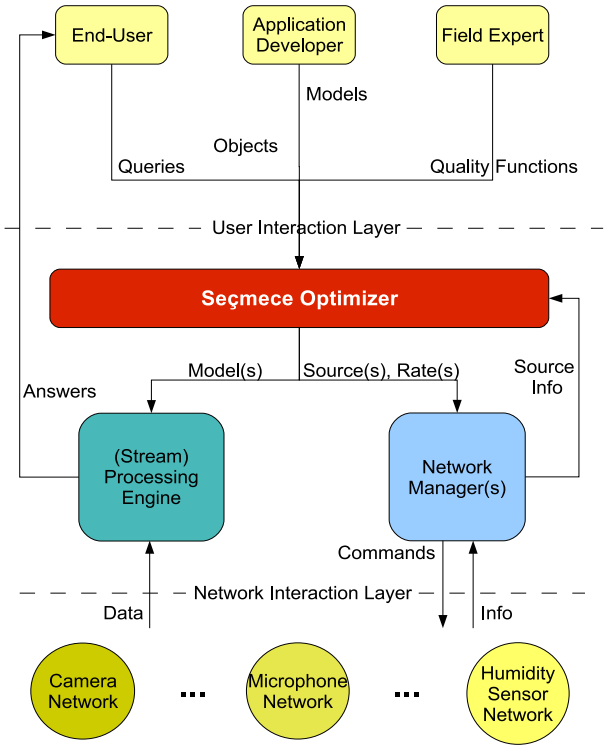

Figure 1: System view

- Project Manager Mariya wants to learn how many hours per day her project members work when it is close to the deadline.

- Employee Sasha wants to learn what time is the best to go to the gym, depending on the parking space being full and who are there.

The examples above are queries in the system. A ***query*** is a question whose answer is of interest. Formally, a *query* should state which *output type* it wants, at which *quality level*, and on which *object*. For example, in the query of Security Chief Matt above, the output type is the photo of the robber, quality is probably defined on resolution, and the object is the parking deck. We assume the queries may require quality information about all the parts of the object being optimized, thus all parts of the parking deck should be observed with some minimum quality.

An ***object*** has a *boundary*, and can intersect with other objects. Seçmece optimization is done independently for each object. For queries which are on intersecting objects, the intersection can be optimized for both queries, and then remaining parts can be optimized independently, considering sources on top of the ones already selected for the intersection.

Queries can be answered by an ***alternative model (AM) set***, which consists of models that can be used interchangeably. Alternative models produce the same type of output, and thus can be used instead of each other, resulting in model redundancy. As an example, one can answer Sasha's gym query in at least these five possible ways (models):

1. Using *infrared beam sensors* to detect a car parking, and *id-card readers* to detect the identity.

2. Using *magnetometers* to detect parking, and *id-card readers* for identity detection.

3. Using *infrared beam sensors* to detect a car parking, and *cameras* to find the identity.

4. Using *magnetometers* for parking, and *cameras* for finding the identity.

5. Using *cameras* for both detecting a car parking, and the identity.

A ***model*** is a way to compute something. It has some number of *input types*, one *output type*, a *model function* from those input types to the output type, and a *quality function* which maps the given input qualities to the output quality. Even models taking the same inputs at the same input quality can have different output qualities. As an example, consider two models for face detection working on the same camera image. One might produce better results than the other, because of the algorithmic differences in the *model function*, where the computation is done. Models with the same output schema are considered *alternative models* and are used to answer queries asking for that output type. For now, we restrict ourselves to the case where input types of a model can only be source types; they cannot be the output types of other models. Nevertheless, the model requiring the output of another model can be expressed as one combined model. The input types of the combined model will be the union of the source types required by those two models, and the output type will be the output type of the model which used input from the other one. Once the models and objects are defined, the end user should only indicate which output type (i.e. alternative model set) it wants on which object, at which quality threshold. Supplying just a threshold does not necessarily make sense. But the details of user interaction are outside the scope of this paper. The basic system view can be seen in Figure 1.

2.3 Quality Functions

In the previous section, we briefly mentioned the quality functions of models mapping input quality to output quality. Quality is a combined measure of sources' sensing qualities, sensing rates (i.e. data latencies), and model specific computational effects (ex. algorithmic differences when performing face detection).

There are multiple functions in the system that enables quality-based optimization, as seen in Figure 2. The first one is the **source quality function** which takes a *source* of some type and a *point* in the object space as parameters, and returns a quality (interchangeably error) value. This error value represents the fact that the reading will have some associated error with the real value of the phenomenon at that point. This is mainly due to hardware limitations (ex.camera resolution), but it may also depend on environmental conditions. But since those conditions are unknown or unpredictable, this is not captured in our model. If conditions are predictable, they can be incorporated into the *source quality function*.

The next quality function is the **input quality function**. The input quality function takes the errors of sources of one

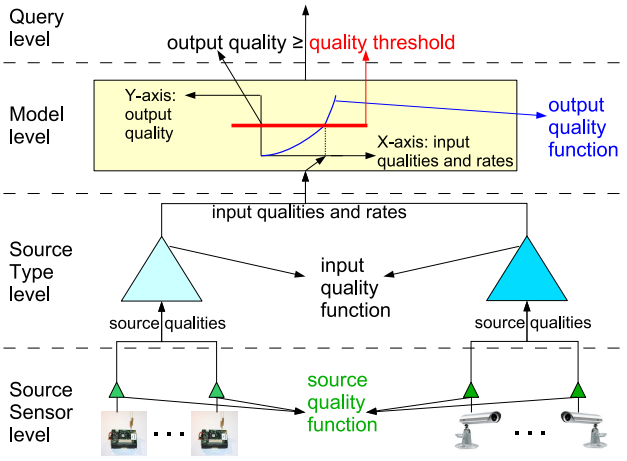

Figure 2: Quality Functions

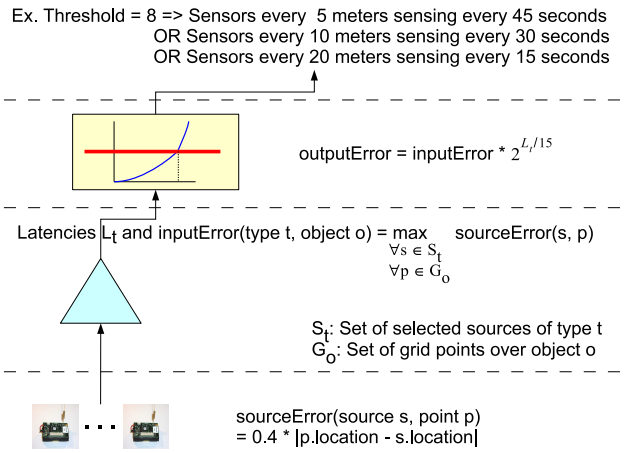

Figure 3: Quality Function for Intel Lab Example

type, and returns an aggregate input quality value. By default, we take the maximum quality among all the sources for each point, and then the overall quality is the minimum quality over all the points in the object space, which somehow guarantees even the worst information is good. Alternatively, any function can be put in place.

The two functions above depend only on the source type, therefore they can be defined by experts in related fields, just once for our system. Then, the application developers are expected to supply a quality function for every model being defined. The **output quality function** of a model will get the result of the *input quality function* defined above, along with associated input rates, and return an output quality as a result. This function combines the qualities of different types of sources used by the model. One important restriction to the output quality functions is that no model getting more input should produce less quality answer.

Intel Lab Example: Here is an example on how to use quality functions. In the Intel Lab setup [1], we have a fire detection query which employs a model using temperature sensors. We would like to select which sensors to use among about 50 available. In our work with real Intel Lab data, we observed that a temperature sensor has an error of $\pm 0.2' \circ C$ per meter. When we take many of these sensors, their quality needs to be combined. We take the quality of a point in the lab as the quality of the closest sensor observing that point. We take the worst quality over all the lab space as the input quality to the model. Then the model output quality function also incorporates the latency effect. Figure 3 shows those quality functions and gives an example quality threshold stating what it would correspond to in our optimization system.

Medical Example: The medical setup [29] has 13 different types of bio-sensors, with one sensor of each type, on the person. The example here shows two models on the thermal state of a person. One model uses only the skin temperature data, which has $\pm 0.2' \circ C$ measurement error, and the other model uses this skin temperature data in combination with the heart rate data, which has ± 5 beats per minute error. In Figure 4, the model on the left is the first one using only the skin temperature, and the model on the right

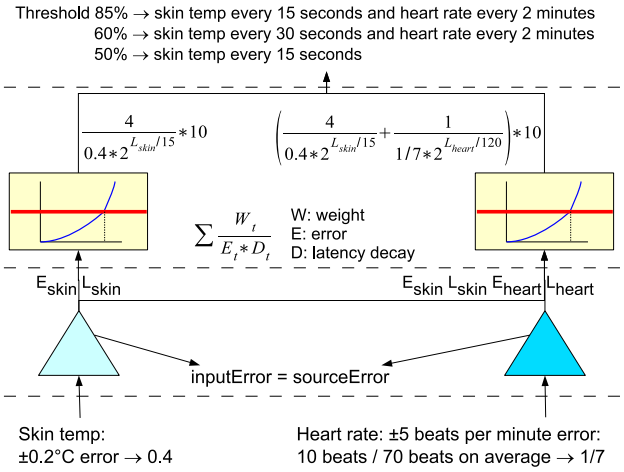

Figure 4: Quality Function for Medical Example

is the other. Since these bio-sensors are not sensing an area, they do not have 'per meter' errors, their errors are fixed. Furthermore, since we have only one sensor per type, the input error at the source type level is the same as the source error. Finally, both the general form of the model output quality function and specific functions are given. The constants in those functions are to help the overall quality to be represented as a confidence value.

2.4 Limitations

Our quality model has some limitations due to the *input quality function* at the source type level. This function provides aggregate information about the sources, and not their exact locations and individual qualities. Furthermore, it depends only on the source type, and not the model using that information. But, some output quality functions require complete source information to compute some specific qualities. We can push such functions to the source type level, but then all models have to use those functions. This unfortunately means that our framework cannot sup-

Aggregate Source Information and Same Rate for Sensors of the Same Type	All Source Information Required Exactly OR Specific Scheduling of Each Sensor
Probabilistic [8, 25, 9, 27] Generalizable [28, 30] Bio-sensors [29]	Exact covered area [25] Application specific [17] Specific schedule [20]

Table 1: Different Output Quality Functions. Our model can hadle the ones on the left column.

port such quality functions. Moreover, because we select only one rate per source type, specific timing schedules requiring different rates for every single source cannot be optimized.

On the other hand, our current quality definition eases defining the quality functions (separation of duty between expert users and end users), and eases the NP-Hard optimization. Yet, it is still applicable to many applications which make use of an aggregate quality value without requiring detailed knowledge about specific sources and hence can be optimized by our system. They include probabilistic calculations and coverage issues, data independent generalizations, and bio-sensor applications, as summarized in Table 1. In the Future Work section, we talk more about extending our approach to other applications.

3. SEÇMECE

3.1 Optimization Problem

Seçmece optimizer's goal is to select models, sources and associated rates to satisfy the quality requirements with the lowest possible cost achieving the maximum system lifetime. The hardness of the optimization comes from the multiplicity of all the options, namely queries, models, sources and rates, and the fact that these optimizations are not independent. For instance, different models can use the same source types, but to satisfy the quality requirements, one model may need many more sensor devices than the other, or a much higher rate. It is practically impossible to try all the possible combinations and pick the best. First, it is computationally too much time consuming, and second it is based on history and predictions, hence it is impossible to make the best choice. Our contribution here is to come up with some solution mechanisms and heuristics for the following NP-Hard *momentary optimization problem*, which minimizes the cost of the system at a given moment, and its application to the system lifetime optimization.

Suppose there are n queries in the system. Each query uses an alternative model set:

$$AM_i = \{M_{ij}\}, i : 1..n, j \geq 1$$

Let M represent the set of all models in our system, and the set S represent all the sources. Our goal is to choose the minimum cost (in terms of battery consumption) subset SS of S with their associated rates, and subset MM of M, such that we choose one model for each query, and pick sources to answer them, satisfying quality requirements. Formally, the momentary cost optimization problem, optimizing a given system instance, is:

$$\underset{\substack{SS\subseteq S \\ rate(s),\forall s\in SS}}{argmin} \quad \sum_{s\in SS} cost(s) * rate(s)$$

subject to

$$\forall i \exists j, \quad M_{ij} \in MM, \quad MM \subseteq M$$
$$\forall M_{ij} \in MM \qquad \exists S_{M_{ij}} \subseteq SS,$$
$$quality_{M_{ij}}(S_{M_{ij}}, rate(S_{M_{ij}})) \geq threshold_i$$

This problem is strongly NP-Hard since *(Minimum) Vertex Cover* problem [14] can be reduced to a very simple version of this problem. Given a graph, (Minimum) Vertex Cover problem is to find a (minimum cost) set of vertices such that every edge in the graph has an endpoint vertex in that set. The reduction to our problem maps vertices of the graph to source types, with only one source sensor each. For each edge, there exists an AM which has two models each with one input type corresponding to one endpoint vertex. Sensor costs are assigned according to the weights in the graph (all equal if unweighted). We do not even care about rates or qualities. After the reduction, selecting one model -with one input- out of every AM means selecting one endpoint of every edge, with the lowest possible total cost.

3.2 System Lifetime

As an even harder problem, we are interested in the system lifetime, instead of just momentary costs as above. The system lifetime is not the same as the network lifetime in sensor networks; the time until the first node dies or the first network partition occurs. ***System lifetime*** means the time the system can continue satisfying the quality requirements of all the queries. While doing that, with every momentary re-optimization, the system can choose different sources and rates, and use different models at different time intervals. Even though Seçmece is designed to optimize the system at an instance, optimizing for the above momentary cost optimization problem, it also considers the following dimensions to maximize the system lifetime:

Exploiting Spatial Data Redundancy using Intelligent Source Selection: An intelligent source selection algorithm should pick fewer sources to achieve the required quality, by exploiting the *spatial data redundancy*. Our approach in Section 3.4.1 performs much better than naive source selection schemes, which do not consider overlapping contributions of sources to the overall quality.

Exploiting Temporal Data Redundancy using Intra-Query Sharing: Making use of *temporal data redundancy* means performing rate selection for source, at the least. But further optimizations are possible with the realization that important source types might be used at slower rates in exchange of using other source types at higher rates. Our *Rate Heuristic* in Section 3.5 makes use of the ***intra-query sharing***, which is the sharing of the source types among models of the same query. Thus, if we can use those shared source types slower, by sacrificing some more consumption on the unshared ones, the lifetime will be extended, since we can continue answering the query even if some model becomes unusable.

Exploiting Model Redundancy using Inter-Query Sharing: *Model redundancy* suggests performing model selection. We can make use of ***inter-query sharing***, sharing of

source types among models of different queries, while performing model selection. Such sharing can be exploited to answer all the queries by consuming collectively less total energy. Results show the importance of exploiting the inter-query sharing, which cannot be taken into account by the *Independent Query Optimization* method in Section 4.1, but exploited by our *Holistic* approach in Section 3.5.

3.3 Assumptions and Clarifications

There are several simplifying assumptions we made while solving this NP-Hard optimization problem. First, we assume that the rates of all sensors of a the same type are equal, making it easier to optimize, while still supporting many real applications. Second, we are not taking into account the networking issues, focusing mainly on sensing. It is the network managers' job to handle networking, hence we assume the network is reliable. In unreliable cases, the quality thresholds might be increased to force our system select more sources, thus trying to increase the reliability. Third, we are not dealing with the directed sensors like cameras. Currently, our optimization system supports only undirected sensors like temperature, humidity, smoke sensors, magnetometers, bio-sensors, etc. The applicability of our system to directed sensors and taking into account networking are further discussed in Section 5.2. Forth, the optimization we perform is data agnostic. Thus, the real quality might be higher or lower than the estimated quality, and the usefulness of the estimate depends on the design of quality functions supplied by the users. Since we cannot predict data in an event detection system, we have to be data agnostic. Fifth, we are also unaware of how the model computation is performed. We will only select which sources should send data to the (stream) processing engine at which rate, but the models are free to use that data in any way they like. The details like model scheduling and windows are outside the scope of the optimization part.

3.4 Per Model Approach

Even though our main optimization approach is the *Holistic* approach, we will start by defining the *Per Model* approach to make it more understandable. The difference between the two approaches is mainly on the way the *Holistic* approach optimizes everything together, instead of a per model approach as in this one. This method optimizes all the source types used by a given model together to achieve a combined quality. This is due to the fact that the quality function of a model depends on all the source types it is using, and cannot be partialized per source type. Before going into the algorithm, we should understand some subproblems and their solutions.

3.4.1 Picking The Next "Best" Source

As an independent subproblem, picking the next "best" source can be described as picking *an additional source* of the given kind to increase the quality of the models, by getting a *better quality* input from *more parts* of the object being optimized. Even this subproblem is NP-Hard, and a similar approach to a similar problem is shown to be a good approximation [3]. The selection is done by iteratively calculating the source utilities and picking the one with the best utility, taking into account the *spatial data redundancy* as explained below.

Consider the object space being optimized as a pure white picture. Each source has a quality function which can return us a quality value for any point in this picture (the *source quality function* defined in Section 2.3). Consider the highest quality as black, and the lowest quality as white. So, if we take a source, it will color the picture with some gradient filling, black at the center, getting gray with distance. Here, the utility is defined in terms of the *input quality function*. For our default case, the combination of the utilities will assign each pixel the blackest value given by the selected sources. Then, the utility of a source is defined as the *blackness difference of the whitest point* between the two pictures; the one before adding that source, and the one after. The reasoning behind this selection strategy is to be able to *decrease the maximum error* in our readings, and to *have a good coverage* of the object. If two sources offer the same utility, the total blackness difference of the two pictures are considered, and the one with higher difference is taken to increase the coverage. After picking the next "best" source, recalculation of the utilities is necessary, since the picture has changed. The selected source paints the current picture, and the other ones will now be considered on top of this new picture. A sample selection visualizing the qualities can be seen in Figure 5.

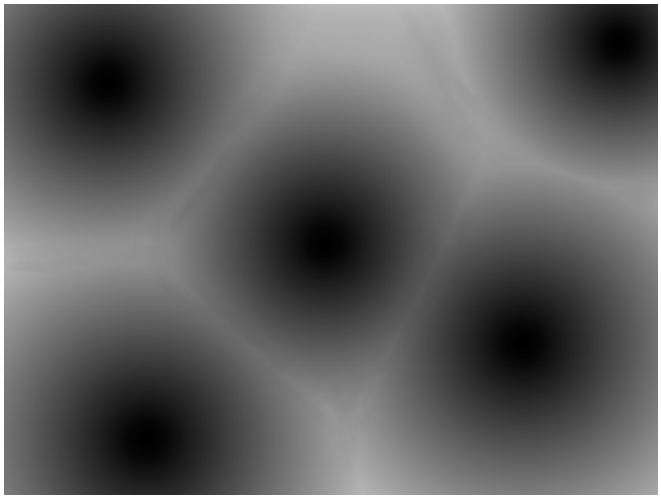

Figure 5: Combined qualities of 5 sources over an object. Note that pure black is the highest quality.

3.4.2 Which Type of Source to Pick Next?

The above strategy is for picking sources of a given type. But we need to decide which type of source we should pick next, by predicting which type is going to be the most useful. For that purpose, we first pretend picking a next "best" source from each source type. For each type, its utility is then the maximum quality improvement percentage of the models by taking that one more source, divided by its cost. Each source type is considered independently, meaning that the quality improvement is realized by just one more source of that type, and nothing else. After computing the utilities of all source types, we actually pick the next "best" source from the type with the best utility. This selection will affect upcoming steps, and so utilities are going to be recomputed.

3.4.3 Source Selection

Given a model, we pick the next "best" source from the

most useful source type, until the model is satisfied. This phase is done for different rate levels, starting from the highest, requiring the smallest number of sources, going to the lowest, using more sources whenever necessary. Even the optimal rate selection subproblem is NP, thus such simplifications help actually solving such a complex optimization problem as ours.

Algorithm 1 Source and Rate Selection

```
for rate from the highest rate to the lowest rate do
    repeat
        Pick the next "best" source from the most useful source type (Sections 3.4.1 and 3.4.2)
    until the model is satisfied
Choose the least costly rate-sources selection
```

The first phase of the *Per Model* approach is composed of running Algorithm 1 for each model separately. At the end of this first phase, we have which sources to use from each type, and at which rate, for each model. As an intermediate phase, we unite the sources required by the least costly model of each AM using that type, and take the maximum rate required by those models. This is an overestimation of the cost, but it guarantees that all these models will have enough quality. Then, we perform the second phase as in the next section.

3.4.4 Model Selection

Given the selected sources of each type, and their associated rates, we choose which models to use with a greedy approach. When choosing models, the system should try picking models with most overlapping sources with each other to increase the system lifetime. This strategy employs *the inter-query sharing*.

Algorithm 2 Model Selection

```
repeat
    Pick the source type with the best utility (see the example below)
    Pick the cheapest models of all AMs using that type
    Remove that source type and AMs from the list
until all AMs are satisfied
```

To illustrate Algorithm 2, consider the following alternative model sets A, B, C, D, E with each subset being a set of input types (represented by numbers) to a model in that AM:

$A = \{ \{1, 11\}, \{5, 8\} \}$
$B = \{ \{3\}, \{7\}, \{17\} \}$
$C = \{ \{4, 5, 11\}, \{9, 12, 13\}, \{9, 14\} \}$
$D = \{ \{12\}, \{14\}, \{17\} \}$
$E = \{ \{11\} \}$

This means, AM A has 2 models with 2 input types each. Only for simplicity here, assume the costs of all source types are 1. Now, for each source type, we will list the least costly model of each AM it supports, and other necessary sources, if any, for completely supporting that AM (for example, source type 9 supports only AM C, and the least costly model using 9 also requires using 14):

1	$A, 11$		
3	B		
4	$C, 5, 11$		
5	$A, 8$	$C, 4, 11$	
7	B		
8	$A, 5$		
9	$C, 14$		
11	$A, 1$	$C, 4, 5$	E
12	$C, 9, 13$	D	
13	$C, 9, 12$		
14	$C, 9$	D	
17	B	D	

The weight of a source type is then the total number of AMs it supports divided by the total cost required (including the cost of the helper sources required to completely support those AMs). For example, the weight of source 11 is 3/4. At this point, we take the heaviest source, which is 17 in our example with weight 2. Thus, our source set becomes $\{17\}$ and our supported AMs set becomes $\{B, D\}$. Then we remove $17, B$ and D from wherever they appear on our list, and the new list becomes:

1	$A, 11$		
4	$C, 5, 11$		
5	$A, 8$	$C, 4, 11$	
8	$A, 5$		
9	$C, 14$		
11	$A, 1$	$C, 4, 5$	E
12	$C, 9, 13$		
13	$C, 9, 12$		
14	$C, 9$		

Now we re-calculate the weights and get the heaviest source, which is now 11 with weight 3/4. 11 completely supports only E, so our new source set becomes $\{17, 11\}$ and our supported AMs set becomes $\{B, D, E\}$. After removing 11 and E, we take 1 and A next, and after that, we take 4 and 5 or 9 and 14 for C (we encourage the reader to continue the algorithm to see these steps). At this point we have all our AMs supported, therefore the algorithm terminates, finding us the best possible result for this example. Note that for the algorithm to give good results, the costs should be realistic.

3.5 Holistic Approach

Having observed that **all the selections being performed affect one another**, we extended our *Per Model* approach to optimize everything together. This complicates the problem but improves the solution greatly. The basic idea is similar to the *Per Model* approach. We perform source selection as defined, but now the utility of a source type in Section 3.4.2 does not depend on only one model, but all models. The biggest difference of this approach is, there is no separate phase for selecting models. While taking sources, we also pick models satisfied by those sources, until we satisfy each query by one model. Again, all these are performed for different rates, trying to get the minimum total cost.

One important issue is that while running Algorithm 3 for different rates, the selected models and sources can be completely different, depending on their costs and the quality requirements, since every time we start with all queries marked as unsatisfied and all sources unselected.

The algorithm above treats all source types equally tem-

Algorithm 3 Holistic Approach - Basic Rate Selection

```
1: for rate from the highest rate to the lowest rate do
2:    Mark all queries as unsatisfied
3:    Clear all selected sources
4:    repeat
5:       Calculate source type utilities (Section 3.4.2)
6:       Pick the next "best" source from the source type
         with the best utility (Section 3.4.1)
7:       if an unsatisfied query is satisfied now then
8:          Pick the satisfying model
9:          Pick the sources that model uses
10:         Mark that query as satisfied
11:   until all queries are satisfied
12: Choose the least costly rate-sources selection
```

porarily. When the algorithm finishes, decreasing the rates of all the sources at once is not possible, but we can try lowering the rates individually in some order. The *Rate Heuristic* in Algorithm 4 exploits the *temporal data redundancy* using the *intra-query sharing*. The source type shared the most among the models of a query, taking the maximum percentage over all queries, has the maximum utility.

Algorithm 4 Holistic Approach - Rate Heuristic

```
1: Calculate rate utilities of source types
2: repeat
3:    Lower the rate of the source type with the maximum
      utility and select sources if necessary as in Algorithm
      3 (without the rate loop)
4:    if all queries can be satisfied then
5:       Lower the utility of that source type
6:    else {cannot slow down that source type}
7:       Set the utility of that source type to 0
8:       Restore the old rate of that source type
9: until all utilities are 0 {no more room for improvement}
```

Remember that the *Rate Heuristic* runs after Algorithm 3. The combination of these two algorithms will be our standard "*Holistic*" approach, with modifications below in Section 4.1 done on it. The run time of this approach is proportional to the square of the number of sources, the square of the number of types, and the number of models in the system.

4. EXPERIMENTS

4.1 Analyzed Approaches and Effects

To understand the advantage gained by exploiting various redundancies in our *Holistic* approach, we modified one part of it at a time and experimented to see the effect.

Grid Source Selection: As explained before, our source selection technique exploits the *spatial data redundancy*. To see the importance, we compared it to a *Grid Source Selection* technique which does not take into account overlapping contributions of the sources to the quality. It changes the picking the next "best" source part in Section 3.4.1 by dividing the object being optimized into grids and picking sources closest to each grid point. The number of grid points is increased whenever the quality is insufficient. Note that the grid shape is also trying to minimize the error given by our default *input quality function*, which makes it not a very naive method.

Basic Rate Selection: We call running only the first phase of the Holistic approach, without rate heuristic, the *Basic Rate Selection*. This algorithm treats all source types equally. We compare it with the standard *Holistic* approach, which exploits *temporal data redundancy* at the *Rate Heuristic* part, considering the (*intra-query sharing*).

Independent Query Optimization: Consider applying the Holistic approach to one query at a time instead of applying it to all queries at once, hence obtaining independent optimizations for each query. This way, the optimization will not be able to take advantage of the *model redundancy* by using the (*inter-query sharing*).

Momentarily Optimal Solution: This optimal solution to the *momentary cost optimization problem* in Section 3.1 is NP-Hard, taking exponential time. It is very important to note that *"Momentarily Optimal"* solution **does not necessarily give the optimal system lifetime**. But, showing the results of optimal lifetime would be too much time consuming considering the fact that even momentarily optimal solutions for small-scale deployments can take days.

4.2 Setups

We have two real setups and hundreds of randomly generated setups for comparing the approaches above. All setups have basic versions for comparison against the *Momentarily Optimal* solution and complex versions with more sources, models and queries for comparison between our modified approaches.

Intel Lab Setup: The basic Intel Lab setup [1] is made up of about 20 temperature sensors, and one query with one model running on top of these sensors. The complex setup for Intel Lab is composed of some 150 sensors, one third sensing *temperature*, one third sensing *light*, and one third sensing *humidity*. It has 2 queries over fire detection and workspace quality control. One query has 3 models, each using one source type, and the other has 2 models, requiring either temperature and light sensors, or temperature and humidity sensors.

Medical Setup: The medical setup [29] we stated before is now our experimental setup. The basic version uses 10 different source types, with one sensor each. It has 4 queries measuring the thermal state, the hydration state, the cognitive state and the life signs of a person. Each query can be answered with at least 2 different models using different -possibly more than one- source types. The complex version has 13 source types, again with one sensor each, and at least 5 models per query.

Random Generated Setups: For further analyzing the performance of our optimization system in other scenarios, we created random setups with all the basic complicating properties we wanted: many sources of different types and multiple queries that can be answered using multiple models. Again, we have both basic versions with very few multiplicities, and complex versions with hundreds of sensors and tens of model-query combinations. Furthermore, the density of sources and quality requirements of the queries are all random, giving us the chance to have setups of all kinds, even ones at the extremes.

4.3 Results

In all the comparisons below, we are comparing other ap-

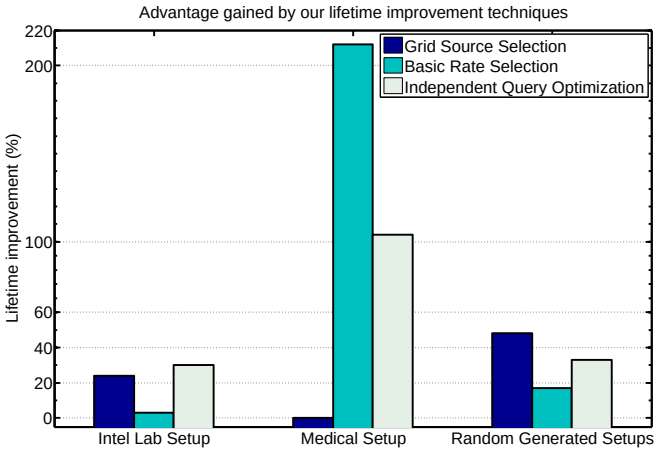

Figure 6: Effect of exploiting different redundancy types on various setups

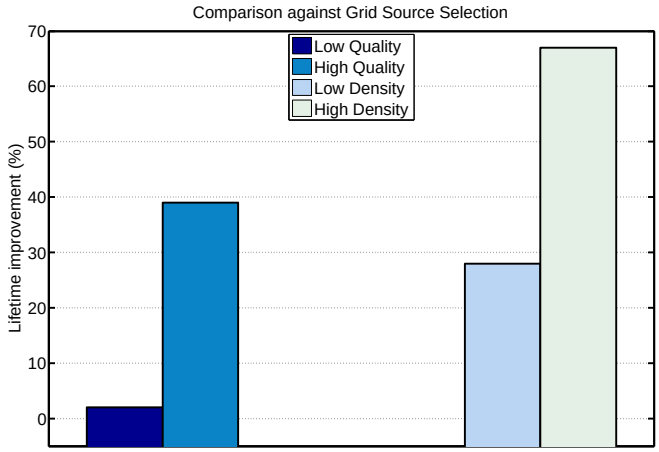

Figure 7: Effect of exploiting Spatial Data Redundancy with varying query quality requirements and source densities

proaches to the *Holistic* approach as defined in Section 3.5, including the *Rate Heuristic*. To find the system lifetime observed using a strategy, we optimize an instance of the system with that method, and use that selection until it becomes useless. A selection of models, sources and their rates becomes useless as soon as one selected source dies (because its battery is drained). Whenever a selection becomes useless, a new optimization is performed with the same strategy. Thus, the lifetime is the total time passed until the moment we cannot perform a selection satisfying all the quality requirements.

General Results: Figure 6 shows the comparison of our approaches on different setups. Our *Holistic* approach achieves about $1.25x$ lifetime than that of *Grid Source Selection* in the real Intel lab setup, and about $1.5x$ lifetime in random generated setups. They perform equally well in the Medical setup since there is no source selection there, and thus no *spatial data redundancy* to exploit. The advantage of our *Rate Heuristic* is more than *three times* in the Medical setup, since there is high *intra-query sharing*. In the Intel setup where the sharing is not much, the advantage is not so obvious, but on the average in the random generated setups, it achieves about 20% gain. Similarly, the advantage of optimizing queries together is about 30% in the real Intel setup and random generated setups, whereas it is *twice* more effective in the real Medical setup, due to high *inter-query sharing*. Below, we will analyze individual dimensions in more detail. We do not claim those are the only possible ways of exploiting the data and model redundancies. Instead, we show that, by using them in our solution, we gained more lifetime, and thus suggest that whoever wants to improve the lifetime should exploit them whenever possible.

Comparison against Grid Source Selection: We claimed that an intelligent source selection strategy exploiting the *spatial data redundancy* is necessary for improving the system lifetime. In Figure 7, we show the advantage of our strategy over the *Grid Source Selection* strategy. Two parameters we analyzed were the quality requirements of the queries and the source densities. As seen from the figure, low quality requirements or low source densities mean there is not much redundancy to exploit and any source selection technique can do the job. But, when the quality requirements or the densities of the sources are higher, the selection should reach the high quality requirement with the least number of sources, choosing these sources intelligently among many alternatives. Thus, in such scenarios, the advantage of our approach is even more. This supports our claim that we should exploit the *spatial data redundancy* whenever possible (when there is lots of redundancy).

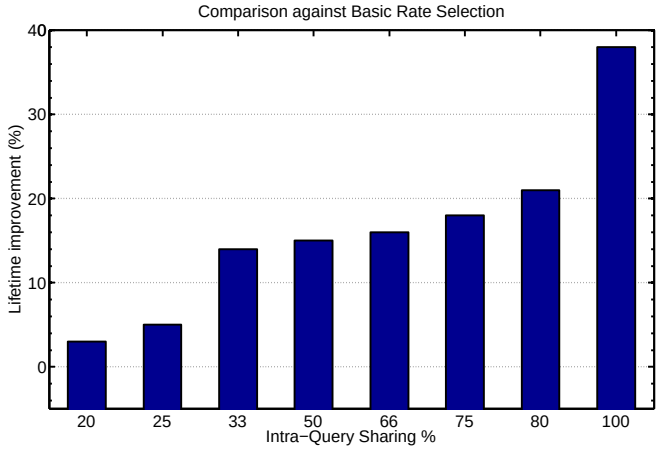

Figure 8: Effect of exploiting Temporal Data Redundancy better by taking into account intra-query sharing

Comparison against Basic Rate Selection: We claimed that to exploit the *temporal data redundancy*, one should make use of the temporal differences between the source types. This means, we should adjust the rates of the sources. A non-trivial way of assigning rates is using the *intra-query sharing*. Sources that are shared by more models of the same query is more important than the others, since if they die, it will not be easy -if not impossible- to satisfy that query. We compared the *Basic Rate Selection* which treats all source types temporarily equally, against the *Holistic* approach including the *Rate Heuristic*, which makes use of this more advanced form of *temporal data redundancy*. The results can be seen in Figure 8.

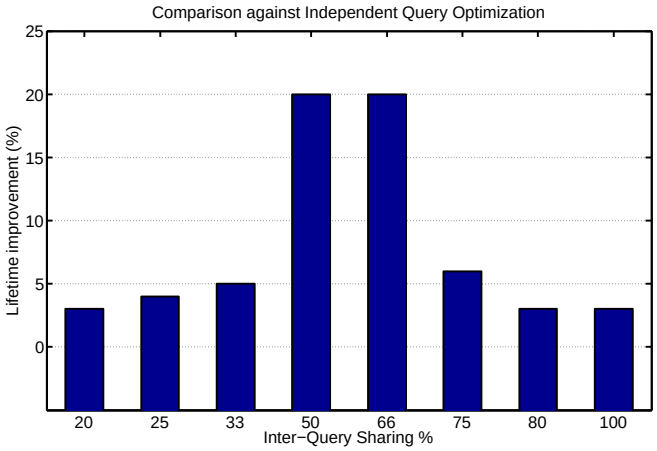

Figure 9: Effect of exploiting Model Redundancy better by taking into account inter-query sharing

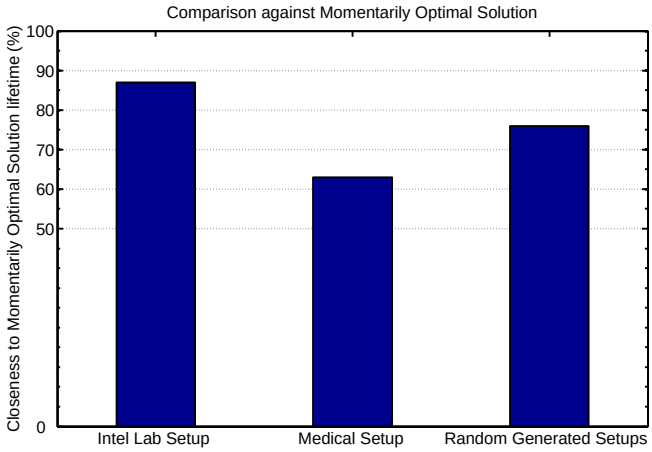

Figure 10: Comparison against Momentarily Optimal Solution

The sharing amount is a rule of thumb that indicates what percentage of models in the query share one source type. We do not claim that this figure covers all possible sharing sets, nor do we claim that this is the only way of exploiting *intra-query sharing*, but the figure does give us an idea. As seen from the figure, the advantage of the *Rate Heuristic* exploiting the *temporal data redundancy* in a better way tends to increase with increasing *intra-query sharing*, which makes it even more important not to treat source types equally in terms of temporal matters.

Comparison against Independent Query Optimization: *Holistic* approach exploits *model redundancy* better by considering the sharing of source types among models of different queries, not just the same query. Even though the *Independent Query Optimization* approach is using essentially the same algorithm, since it does not have the information about all the queries at the same time, it cannot take advantage of the *inter-query sharing*. The effect can be seen in Figure 9. The sharing amount determines what percentage of the sources of each model in one query forms the sources of one model in the other query.

When there is not much sharing, performing independent optimization is almost the same, since there is not much redundancy to exploit. On the other hand, when the sharing is too much, the independent approach again performs almost the same. This is due to the fact that even though it does not consider the sharing, whatever it chooses, there is some redundancy exploited anyways. For example, if all queries required the same quality answer using the same models (100% sharing), there would not be any difference, since the selection done individually for each query would give the same result. The small difference here comes from the fact that queries not always require the same quality thresholds, so the selections might be different. The figure suggests that we should exploit the *model redundancy* whenever possible.

Comparison against Momentarily Optimal Solution: First, remember the fact that this method only gives us the optimal solution for the *momentary cost optimization problem* in Section 3.1. It does not necessarily give the *optimal lifetime*. As Figure 10 shows, our *Holistic* approach achieves only 23% less performance than the exponential-time *Momentarily Optimal* solution. In more than 100 randomly generated setups we experimented, 15% of the time our *Holistic* approach performed equally well, and in another 15% of the cases, we achieved even longer lifetimes.

Since we performed better -in comparison to the Momentarily Optimal solution- in the Intel lab setup than the Medical setup, a possible conclusion from the figure is that our source selection performance is better than our model or rate selection performance. The Intel lab setup is only source selection, no model or complicated rate selection, showing that our source selection strategy performs only 12% worse. In the Medical setup, there is no source selection performed, but only model and rate selection. There, we performed about 36% worse. This hints us that the future work should probably be more focused on model and rate selection, possibly incorporating other means of exploiting *temporal data redundancy* and *model redundancy*.

4.4 Conclusions of the Experiments

In conclusion of the experiments above, we showed that by using intelligent source selection techniques, and by exploiting the inter- and intra-query sharing of source types, or in other words by **exploiting the data and model redundancy** in the system, our *Holistic* approach can help increasing the system lifetime. The setups in consideration are expected to have many sources of different types, and many models for each of the multiple queries, sharing some source types. Thus, when doing optimizations in such an FSN environment, exploiting the redundancies is necessary and useful. The results against *Momentarily Optimal* solution are also promising since we can perform close to or sometimes even better than that exponential-time solution. With this, we also showed that the system lifetime optimization is different than performing only momentary cost optimization.

5. RELATED AND FUTURE WORK

5.1 Related Work

As in Federated Databases, Federated Sensor Networks integrate different sources of different types and capabilities. Even though the underlying system is heterogeneous and complex, the users see a unified system view. A se-

lection of these sources are performed based on some information feedback from the sources. But, although these systems are very similar in those senses, there are major differences in their optimizations. The optimization in Federated Databases try to generate query plans by re-ordering the query operators, choosing which sources to use to answer which parts of the query and deciding whether to execute joins locally or remotely [18]. In our case, we have source, model and rate selection necessary for continuous queries. The quality-based optimization is also an important issue, with the quality being completely redefined in our context. The resource limitation is mainly the battery power. The goal can be minimizing the cost of momentary selections, or more importantly the system lifetime. Thus, in such a Federated Sensor Network system, optimization is a novel and very important piece.

As the current state of art in sensor networks, there are some widely accepted approaches for cost reduction. One is in-network computation, as in TAG [22]. Another is using probabilistic approaches and predictions [13] to minimize communication costs. There is also work focusing on sensing area coverage [4] using sampling strategies. But to the best of our knowledge, Seçmece is currently the only comprehensive optimization technique for complex FSN applications selecting models, sources and rates all together. For parts of this optimization problem though, there are similar works.

Sensor selection problem is widely analyzed in sensor networks. There are different techniques, trying to minimize the cost [26], improve the network lifetime [15] (instead of a general system lifetime definition as in our case), or fulfill some utility [5] or coverage [10]. These techniques show similarities with our source selection approach (picking the next "best" source), as explained below.

Source selection part has similarities with what has been done in data fusion for web information sources. Similarly, those works try to select data sources by looking at their coverage and overlap [19]. Their density functions [24] can be thought as corresponding to our *source quality functions*. Query dependent densities in that area are just the *output quality functions* of our models. Our difference is coverage (and so overlap) in picking the next "best" source is defined as gradient fillings, instead of a boolean manner. A sensor's coverage is not same at all points [21], and thus the overlap can have different effects. Moreover, instead of checking overlap of only two sources, Seçmece is checking for any overlap, leading to a more accurate selection.

Another similar area in source selection again is sensor selection using probabilistic models. The BBQ approach [13] uses some probabilistic models to predict sensor data. If the model confidence is not enough, it uses a pull-based mechanism to obtain data from more sensors, which increases latency. Also the model makes it very hard to detect outliers, since real sensor data is not used. A more recent work [11] uses spatial correlations between sensors and probabilistic methods for predictions. Its important difference from BBQ is now this approach is push-based. Unfortunately, both of these works are query-specific; they can handle only some types of queries well. On the contrary, Seçmece optimization is general, and it can even capture such correlations if *source quality functions* are designed accordingly. One more difference of our work is unselected sensors can even stop sensing, if routing permits, whereas that work required all sensors to sense, trying to minimize only the communication costs. Such an optimization is important especially for applications using high sensing-cost sensors like battery-operated cameras.

Finally, the selection problems here can be related to task allocation in multi-robot systems, if we think of sensors as robots with different capabilities. The strong NP-Hardness of this problem is then described in such a formal analysis [16].

5.2 Future Work

Very briefly, an important extension is adding directed sensor support to our system, considering the direction of sensing. Next, the model should be changed to come over the limitations on the quality functions. The *input quality function* should be removed, and the source qualities should directly be input to the *output quality function* of a model. Having done that, it also requires changing the picking the next "best" source part so that we now consider the quality improvements per model. This sacrifices complexity of both optimization and definition in favor of a more general framework.

A further extension can be adding lifetime constraints, possibly in combination with some quality constraints. We can start with a 'guess' of quality thresholds and optimize accordingly. If we are short of the required lifetime, we decrease the quality thresholds, otherwise we can try increasing them. How to 'guess' the qualities, how to increment/decrement them, in which order, etc. remain as areas requiring further research.

Another important issue is experimenting with real sensors, seeing how often reoptimization needs to be done and how costly the reconfiguration is. At this point, networking issues can be integrated into the optimization, possibly taking into account the remaining battery powers.

Adding actuation support to the system will likely to be a challenge, because the system should know about actuators' capabilities, and should have some means to control them. No current approach goes into details of actuation, but they will be a very important part of FSN applications in the near future. For example, consider a speech processing application which needs to use images and sounds in a synchronized way to obtain higher quality results. The camera and microphone positioning and directions (if mobile), and possibly multi-action scheduling would be crucial to the system, which will also be part of a more sophisticated selection problem. As other examples, we can also consider robotics applications that incorporate many sensing devices and many actuators.

6. CONCLUSION

In conclusion, it is envisioned that in large-scale sensor networks applications, many sensors of different kinds would be deployed and used for many tasks simultaneously. These tasks can be accomplished by multiple models, achieving different qualities for various costs. We have explored many such applications.

Considering such a deployment, the user should be freed from the burden of dealing with the heterogeneity issues and the efficiency of the system. We claimed that using a ***Federated Sensor Networks*** system can ease the development of those applications, and can take care of the optimizations of critical resources like battery. We showed that this

optimization problem is strongly NP-Hard, because of the multiplicities of the sources, rates, models and tasks, and because of the inter-dependency of their optimization. Furthermore, we extended the optimization problem with the ***system lifetime***, the time the system can continue satisfying the quality requirements of the queries.

Seçmece is a quality-based optimization system for FSN applications, selecting which models to execute on which sources at which rate, satisfying the quality requirements of the queries with low total cost, extending the system lifetime. Our results on both real and random generated setups supported our claim that exploiting the ***data and model redundancies*** in the system by performing intelligent source selection and taking into account ***inter- and intra-query sharing*** help improving the system lifetime and should be employed whenever possible. Moreover, we showed that finding the "momentarily optimal" solution for the momentary cost optimization problem is not enough to achieve optimal system lifetime. Our solution is proved to be promising by performing close to, and in some cases even better than, the exponential-time momentarily optimal solution.

7. REFERENCES

[1] Intel lab dataset. *http://berkeley.intel-research.net/labdata/*.

[2] A. Baggio. Wireless sensor networks in precision agriculture. *REALWSN*, 2005.

[3] M. Bagheri, M. Hefeeda, and H. Ahmadi. A near optimal k-coverage algorithm for large-scale sensor networks. *Simon Fraser University Technical Report TR 2006-10*.

[4] B. A. Bash, J. W. Byers, and J. Considine. Approximately uniform random sampling in sensor networks. *DMSN*, 2004.

[5] F. Bian, D. Kempe, and R. Govindan. Utility-based sensor selection. *IPSN*, 2006.

[6] J. Biswas, S. Das, Q. Qiu, V. Chava, and P. Thang. Quality aware elderly people monitoring using ultrasonic sensors. *ICOST*, 2005.

[7] J. Biswas, P. Yap, V. Foo, Q. Qiu, A. Aung, P. Thang, and Q. Guopei. Use of pervasive monitoring technology as compared to direct observational methods using the soapd scale in the measurement of agitation in patients with dementia. *Institute for Infocomm Research (I2R) and Alexandra Hospital*, 2005.

[8] T. E. Boult, R. C. Johnson, T. Pietre, R. Woodworth, and T. Zhang. A decade of networked intelligent video surveillance. *ACM Workshop on Distributed Camera Network*, 2006.

[9] Q. Cao, T. Yan, J. Stankovic, and T. Abdelzaher. Analysis of target detection performance for wireless sensor networks. *DCOSS*, 2005.

[10] W. Choi, S. K. Das, and H. J. Choe. Constrained random sensor selection for application-specific data gathering wireless sensor networks. *EmNets*, 2005.

[11] D. Chu, A. Deshpande, J. M. Hellerstein, and W. Hong. Approximate data collection in sensor networks using probabilistic models. *ICDE*, 2006.

[12] M. Connolly and F. O'Reilly. Sensor networks and the food industry. *REALWSN*, 2005.

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

[14] I. Dinur and S. Safra. On the hardness of approximating minimum vertex cover. *Annals of Mathematics*, 162(1):439–486, 2005.

[15] Q. Dong. Maximizing system lifetime in wireless sensor networks. *IPSN*, 2005.

[16] B. P. Gerkey and M. J. Mataric. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23(9):939–954, September 2004.

[17] C. Gui and P. Mohaparta. Power conservation and quality of surveillance in target tracking sensor networks. *MobiCom*, 2004.

[18] L. Haas and E. Lin. Ibm federated database technology. *IBM Corporation, Armonk, NY*, March 2002.

[19] T. Hernandez and S. Kambhampati. Improving text collection selection with coverage and overlap statistics. *WWW*, 2005.

[20] J. Jeong, S. Sharafkandi, and D. H. C. Du. Energy-aware scheduling with quality of surveillance guarantee in wireless sensor networks. *DIWANS*, 2006.

[21] B. Liu and D. Towsley. A study of the coverage of large-scale sensor networks. *MASS*, 2004.

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

[23] D. Malan, T. Fulford-Jones, M. Welsh, and S. Moulton. Codeblue: An ad hoc sensor network infrastructure for emergency medical care. *WAMES*, 2004.

[24] F. Naumann, J.-C. Freytag, and U. Leser. Completeness of integrated information sources. *Information Systems*, 29:583–615, 2004.

[25] E. Onur, C. Ersoy, and H. Deliç. On the quality of deployment in wireless sensor networks. *ConTel*, 2005.

[26] M. Perillo, Z. Ignjatovic, and W. Heinzelman. An energy conservation method for wireless sensor networks employing a blue noise spatial sampling technique. *IPSN*, 2004.

[27] S. Ren, Q. Li, H. Wang, X. Chen, and X. Zhang. Analyzing object detection quality under probabilistic coverage in sensor networks. *IWQoS*, 2005.

[28] S. Slijepcevic, S. Megerian, and M. Potkonjak. Location errors in wireless embedded sensor networks: Sources, models, and effects on applications. *Mobile Computing and Communications Review*, 6(3).

[29] N. Tatbul, M. Buller, R. Hoyt, S. Mullen, and S. Zdonik. Confidence-based data management for personal area sensor networks. *DMSN*, 2004.

[30] M. Voit and R. Stiefelhagen. Tracking head pose and focus of attention with multiple far-field cameras. *ICMI*, 2006.

[31] G. Werner-Allen, J. Johnson, M. Ruiz, J. Lees, and M. Welsh. Monitoring volcanic eruptions with a wireless sensor network. *EWSN*, 2005.

[32] K. Whitehouse, F. Zhao, and J. Liu. Semantic streams: A framework for composable semantic interpretation of sensor data. *EWSN*, 2006.