# Where in the Sensor Network Should the Join Be Computed, After All?

Mirco Stern, Erik Buchmann, Klemens Böhm

Institute for Program Structures and Data Organization, Universität Karlsruhe (TH), Germany,
Mirco.Stern|Buchmann|Boehm@ipd.uni-karlsruhe.de

**Abstract.** Data acquisition in battery-powered sensor networks must be energy-efficient. Given this objective, we study the following problem: At which network node(s) should a join be computed? One alternative is centralized, i.e., at one location, vs. distributed. The problem with centralized approaches is that they must consolidate the data from the entire network at the particular site. This is expensive. Thus, we investigate when exactly the distributed alternative is more efficient. To do so, we observe that processing the join in a distributed way consists of two high-level tasks, which determine the energy consumption: (1) Derive the information to find the optimal join location(s). (2) Compute the result, given these locations. Understanding how each of these tasks should ideally be performed is not trivial: Analytic models result in non-differentiable formulae. Further, the number of alternative distributions is extremely high. We address these issues by applying statistical methods. Our contribution is to show that the second task should not be distributed, besides exceptional cases. Regarding the first task, we show that information beyond knowing which tuples join does not help to optimize the processing.

## 1 Introduction

Wireless sensor networks proliferate in many application domains, from environmental monitoring to industrial maintenance. They typically consist of battery-powered nodes with constrained computation and communication capabilities. To increase the lifetime of the network, energy-efficient mechanisms for data acquisition are mandatory.

Our focus is on processing the join efficiently. The join is the classical database operator to express relationships and to analyze sensor data.

EXAMPLE 1.1: A meteorologist is interested in analyzing the following unusual observation: At nearby locations there is a significant difference in atmospheric pressure. A query that acquires the necessary sensor readings is expressed in SQL using a join:

```
SELECT A.*, B.*
FROM Sensors A, Sensors B
WHERE distance(A.x, A.y, B.x, B.y) < 100m
AND A.pressure - B.pressure > 0.1hPa
ONCE
```

Here, Relation `Sensors` serves as a database abstraction of the sensor network. Every node is represented by a tuple with one attribute per sensor of that node (e.g.,

temperature, light). `ONCE` specifies that the query refers to the current state of the network ("snapshot query"), cf. Section 3. □

Our goal is to process join queries using a minimal amount of energy. Typically, sensing and communication dominate the power consumption of a node by orders of magnitude [1–3]. As the sensing costs are orthogonal to the join algorithm, we must minimize the communication to process the join energy-efficiently [1].

More specifically, we investigate the problem *at which nodes* of the network the join should be processed. In particular, join queries in sensor networks may join tuples located on arbitrary nodes and may involve arbitrary join conditions *in the general case*.

One alternative for processing the join is centralized, i.e., at a single location. The 'basic' variant of a centralized join performs all the computations at the base station. Centralized approaches are however expensive: They require to transfer each tuple to the processing site which may be far from the nodes involved in the join. We provide a more thorough discussion in Section 4.

In contrast, operators like projection and selection can be efficiently executed by distributing the computation [4, 5]. The idea is to process the data close to the sources to reduce its volume before transmission. This gives way to the following question: Can distributing the join yield a more efficient processing in the general case? So far, distributed join methods are advantageous in specific scenarios only (cf. Section 2).

A distributed join processing must meet two requirements: (a) Any pair of tuples that join must be in the result. (b) The processing sites have to be chosen so that they are close to the data sources. Thus, distributing the join includes two high-level tasks: (1) Set up: Derive the information required to process the join efficiently and correctly in a distributed way. (2) Result computation: Compute the result, given the setup.

The setup is as important as the actual computation of the result: Identifying optimal join locations requires knowledge of the current state of the network. But acquiring this knowledge can be costly. For instance, the optimal location in the centralized case can be derived from the join selectivity and the placement of the nodes (cf. Section 4).

Our goal is to understand which parts of join processing can/ should be distributed for the sake of efficiency. In particular, we seek to derive optimal solutions for both of the high-level tasks. This analysis is difficult, mainly for two reasons: (1) Analytic models of join strategies cannot be reduced to formulae that are differentiable. For our analysis we borrow from solutions of the weighted Fermat problem and the computation of Steiner trees, and we use statistical methods to provide strict confidence bounds for our results. (2) The number of possible distributed join locations is huge, resulting in a combinatorial problem. A key idea of our work is to develop an estimator that provides bounds on the efficiency of distributed joins. Our contributions are as follows:

**Identifying the knowledge necessary to devise optimal join locations.** We identify which knowledge the set up step must gather. Our result is that it is sufficient to know which of the tuples actually join. Further information like the join partners of a tuple or their locations does not help to optimize the processing. This is an important insight regarding the design of concrete join methods: We reduce the problem to finding ways of acquiring this knowledge efficiently.

**Identifying optimal locations for join computation.** We show that, given the knowledge which tuples join, the optimal location for the result computation is the base station

(except for one special case, cf. Section 5.3). This finding restricts the set of distributed join methods. Once we have identified the tuples contributing to the result it is optimal to ship them to the base station and join them there. In other words, any join implementation that performs a pre-filtering step should not distribute the actual join processing.

What do these results mean? We prove that the optimal distributed join algorithm for the general case consists of a (possibly distributed) filtering phase, followed by a centralized result computation. Any energy-savings potential lies in devising an efficient filtering scheme. Alternative join approaches can be optimal in specific scenarios only.

**Paper outline.** Section 2 reviews join processing in sensor networks. Section 3 presents preliminaries of our work. Section 4 discusses centralized approaches. Our analysis of distributed join processing is presented in Section 5. Section 6 concludes.

## 2 Related Work

Energy consumption of operations like projection, selection, [6] and aggregation (e.g., [7, 8]) are well studied. Efficient implementations exist in data-management systems for sensor networks such as TinyDB [4] or Cougar [5]. However, these systems do not support join operations well: TinyDB allows to join data tuples that are located on the same node only. Although [2] argues that an in-network join can be beneficial, Cougar does not feature this. REED [1] allows for a join of a static external relation and sensor data. REED distributes the static relation among groups of adjacent nodes so that each node can access these tuples at little cost.

In the following, centralized and distributed approaches are presented that process joins over several sensor relations inside the network. As the applicability of all of them is limited, they substantiate our central question: Which parts of the join processing can be distributed in order to increase efficiency?

**Centralized join methods.** A number of approaches compute the join at a single location inside the network. [9] studies long-running join queries in sensor networks. The authors reduce the problem to a variant of the task-assignment problem and adaptively relocate the operator. [10] computes the join on the path between the input data and the query issuer. [11] extends the approach for range queries. Coman et al. [12] present the details of computing the join at a central location inside the network.

Centralized in-network approaches are more efficient than joining the tuples at the base station (cf. Section 4) if (1) the join selectivity is very high, and (2) the tuples transmitted to the central site are located close to each other, compared to their distance to the base station. Thus, centralized approaches are not efficient when it comes to general join queries, i.e., if the tuples are distributed arbitrarily in the network.

**Distributed join methods.** Some of the approaches that distribute (parts of) the processing will serve as illustrations throughout this paper. Yu et al. [13] propose an approach which uses a pre-computation. The idea is to construct one synopsis per relation which is used to identify the tuples that join. In addition, the optimal join location is computed for subsets of these tuples, i.e., the result is computed at different locations. However, the applicability of this approach suffers from the same restrictions as the centralized ones. Thus, this approach does not succeed in extending the applicability by means of distribution. The approach by Yiu et al. [14] joins tuples from neighboring

nodes, i.e., the join condition is $distance(A, B) \leq d$ where $d$ is less than the communication range. The idea is that each node of Relation A broadcasts its tuple. Each node of Relation B performs the join and sends the result to the base station. Again, this distributed approach is limited in its applicability to a specific join condition which guarantees that the tuples involved are close to each other. Finally, an approach that is designed for the application of tracking rare events is presented by Yang et al. [15]. The approach is based on a pre-computation in which one of the relations is distributed to serve as a filter. While this approach incorporates a distributed filtering, it cannot serve as a general join method. This is because it requires one of the relations to be small (a few tuples). Finally, Yang et al. compute the final result at the base station. Thus, the question remains which parts of the processing can be distributed to increase efficiency.

**Analysis of join processing.** Coman et al. [16] implement some concrete join strategies and compare them using simulations. Their goal is to find out under which circumstances a particular method is superior. Their analysis encompasses centralized methods as well as a semi-join. In contrast to our work, the paper does not address distributed join processing.

## 3 Preliminaries

This section discusses the design space for join methods and features a problem statement. In addition, we specify our network model and the communication costs.

### 3.1 Join Queries over Sensor Networks

To facilitate queries over sensor networks the network is seen as a *(sensor) relation*. Networks consisting of homogeneous nodes are represented as a single table with one attribute per sensor (e.g., temperature, light) and one tuple per node. If the network is heterogeneous, groups of homogeneous nodes form different relations. We say that *a node belongs to a sensor relation* $R$ if it contributes a tuple $t$ to $R$.

Our analysis refers to join queries with the following general structure:

```
SELECT A.attrs, B.attrs
FROM Relation_1 A, Relation_2 B
WHERE preds(A) AND preds(B)
AND join-exprs(A.join-attrs, B.join-attrs)
ONCE
```

The query covers two sensor relations and a set of join conditions that are arbitrary expressions over the join attributes. In the special case of a self-join, the `FROM` clause contains the same relation twice. Optional predicates in the `WHERE`-clauses can narrow down the scope of the query.

The semantics is the standard SQL semantics extended for temporal aspects of sensor data. In particular, we adopt the non-SQL clause `ONCE` from TinyDB [17]: It specifies that the result is computed based on the current values. In particular, `SELECT * FROM Rel_1 ONCE` returns a single tuple from each node that belongs to `Rel_1`.

### 3.2 Design Space for Join Processing

Our problem is to find an optimal *join strategy*. Intuitively, a strategy states how the join is computed.

DEFINITION 3.1 (Strategy) *A strategy is (a) a set of locations where tuples are joined and (b) the routes along which tuples are sent.*

Before discussing alternatives for join strategies, we introduce two basic requirements that have to hold for any efficient strategy:

REQUIREMENT 1 (Correctness of the Result) *The join result has to be correct.*

Correctness is a particular concern in distributed settings. We have to ensure that every pair of tuples that join meets at (at least) one location.

REQUIREMENT 2 (Minimal routes) *An optimal strategy has to minimize the routes along which the tuples are sent.*

This requirement helps to restrict the set of candidates for optimal strategies. A strategy that sends tuples on an unnecessarily long route has a superior strategy that sends on a shorter route. – Having described the basic requirements, we now discuss the design space of join strategies according to three dimensions: (1) *number of processing sites*, (2) *locations of processing* and the (3) *granularity of knowledge* required to identify the strategy. Figure 1 serves as an illustration. We discuss each of the combinations in this paper including those that are not named explicitly.

**Dimension 1: Number of Processing Sites.** Along the *number of sites* dimension, we make the following distinction:

DEFINITION 3.2 (Centralized Strategy) *A centralized strategy is a strategy that performs the join at a single node.*

DEFINITION 3.3 (Distributed Strategy) *A distributed strategy may consist of multiple ($\geq 1$) join locations.*

Distributed strategies are defined as a generalization of centralized strategies, i.e., each centralized strategy is a distributed one.

According to our definition, centralized strategies perform the join at one location. In practice, resource limitations might require several neighboring nodes to compute a join. However, we want to provide bounds on the communication costs of different strategies. Using a single node will serve as a lower bound for the communication costs of centralized strategies, as we will discuss in Section 4.

**Dimension 2: Processing Locations.** We distinguish between the processing locations *root* and *in-network*.

DEFINITION 3.4 (Root Strategy) *A root strategy transfers all tuples on the shortest path to the base station where the result is computed.*

Root strategies are centralized strategies. We refer to a centralized strategy where the join location is different from the root as *centralized in-network strategy*.

**Dimension 3: Granularity of Knowledge.** Identifying the optimal join strategy requires knowledge of the current state of the network. For instance, it has been shown for centralized strategies that the optimal location depends on the join selectivity and on the placement of the nodes [16]. We incorporate the acquisition of knowledge in

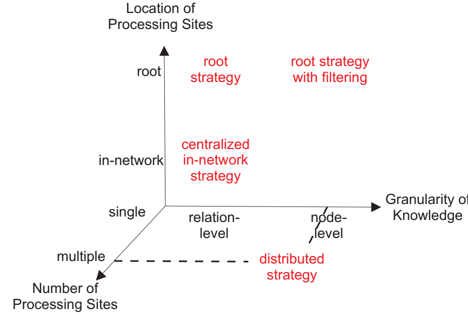


**Fig. 1.** Design space of join strategies

our analysis since the associated costs can vary significantly, depending on the level of detail that is required. In the following, we introduce three such levels.

Intuitively, we can identify an optimal join strategy if we know everything about the state of the network:

DEFINITION 3.5 (Complete Knowledge) *We have complete knowledge of the state of the network if we know the set of sensor relations.*

When knowing the sensor relations we know the location of each node as well as its current sensor readings. Further relevant information can be inferred, e.g., the join partners of a tuple. Thus, complete knowledge would be ideal to identify optimal join strategies. However, acquiring complete knowledge is prohibitively expensive in terms of communication, and it is unnecessary: If we have complete knowledge of the sensor readings, we also know the join result. Thus, we search for abstractions of (parts of) the complete knowledge which suffice to identify an optimal strategy.

DEFINITION 3.6 (Knowledge at the granularity of nodes) *If the knowledge describes the state of a single node, it is at the granularity of nodes.*

EXAMPLE 3.1 Knowing the join partners of a tuple or their locations, or knowing which tuples do not join are examples of knowledge at the granularity of nodes. □

DEFINITION 3.7 (Knowledge at the granularity of relations) *If the knowledge abstracts from single nodes and describes a relation as a whole, it is at the granularity of relations.*

EXAMPLE 3.2 The join selectivity or the number of nodes of a relation are knowledge at the granularity of relations. □

Our analysis will focus on these levels of abstraction. Specifically, we will show in Section 5 that knowledge at the granularity of nodes is a prerequisite to identify an optimal distributed strategy.

### 3.3 Problem Statement

In order to quantify efficiency we introduce the relative measure *gain*. It compares the costs of a strategy ($cost_{strat}$) to the costs of a reference strategy ($cost_{ref}$). We use a relative measure in order to abstract from communication hardware.

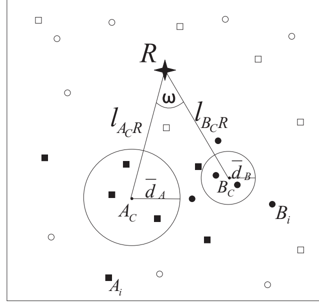


**Fig. 2.** Scenario

| | |
|---|---|
| $n_A, n_B$ | Number of tuples of Relation $A$, $B$ involved in the join |
| $l_{\mathcal{A_C R}}, l_{\mathcal{B_C R}}$ | Distance from center of mass, $\mathcal{A_C}$, $\mathcal{B_C}$ to the root $\mathcal{R}$ |
| $\omega$ | Angle $\sphericalangle \mathcal{A_C R B_C}$ |
| $\bar{d}_A, \bar{d}_B$ | Distribution (mean distance) of nodes with respect to their center of mass |
| $c_a, c_b$ | Relative costs of sending $t_A$, $t_B$ for one hop with respect to $t_{AB}$ |
| $\sigma$ | Join selectivity |

**Table 1.** Overview of the parameterization

| | |
|---|---|
| $t_A, t_B, t_{AB}$ | Tuples of Relation $A$, $B$ and a result |
| $\mathcal{A_C}, \mathcal{B_C}$ | Center of mass of Relation A, B |
| $\mathcal{A}_i, \mathcal{B}_i$ | Node of Relation A, B |
| $\mathcal{R}$ | Root node |

**Table 2.** Further notation

DEFINITION 3.8 (Gain)

$$gain_{ref} = 1 - \frac{cost_{strat}}{cost_{ref}} \tag{1}$$

Modeling the costs of the strategies is different for the centralized and the distributed case and will be discussed in the corresponding sections. In both cases, the root strategy will serve as a reference point. In order to simplify notation, gain refers to gain$_{root}$. According to the preceding definitions the root strategy is an instance of a centralized as well as a decentralized strategy. Thus, the following holds:

COROLLARY 3.1 *For optimal centralized and decentralized strategies,* $cost_{strat} \leq cost_{root}$*. Therefore,* $gain_{root} \in [0, 1]$. □

DEFINITION 3.9 (Optimal Strategy Problem) *Find the strategy that maximizes* $gain_{root}$.

### 3.4 Network Model

In the following, we present and justify our network model. We refer to the two relations to be joined as $A$ and $B$. The relations contain those nodes that result from applying

the selection predicates (WHERE-clause) to a sensor relation. Figure 2 represents the nodes of Relation $A$ as filled squares. They are denoted as $\mathcal{A}_i \in \{\mathcal{A}_1, ..., \mathcal{A}_m\}$. Nodes of Relation $B$ are depicted as filled circles. Non-filled shapes stand for nodes whose tuples are excluded by the WHERE-clause of the query. The root node $\mathcal{R}$ is depicted as a star. We refer to a tuple of Relation $A$ or $B$ as $t_A$ or $t_B$, respectively. $t_{AB}$ represents a tuple that results from joining $t_A$ with $t_B$.

In our analysis we distinguish between different states of the network according to the following parameterization: Relation $A$ can be described by its center of mass $\mathcal{A}_c$, the *mean* distance $\bar{d}_A = \frac{1}{n_A} \cdot \sum_i d(\mathcal{A}_c, \mathcal{A}_i)$ of each node to the center and the number of nodes $n_A$. Relation $B$ can be described in the same way. The relative location of the nodes of $A$ and $B$ to each other and to the root node are described by the angle $\omega = \sphericalangle \mathcal{A}_c \mathcal{R} \mathcal{B}_c$ and the distances $l_{\mathcal{A}_c\mathcal{R}}$, $l_{\mathcal{B}_c\mathcal{R}}$.

Given these parameters, we can abstract from single nodes by describing the relation as $(\mathcal{A}_c, \bar{d}_A, n_A)$. On the other hand, our numerical methods require concrete sets of nodes based on the parameters $(\mathcal{A}_c, \bar{d}_A, n_A)$. We therefore assume:

ASSUMPTION 3.1 *The nodes of a relation are uniformly distributed within ($\mathcal{A}_c, \bar{d}_A$).*

Finally, we will refer to the join selectivity as $\sigma$, which is defined as the ratio of the cardinality of the join result to the input, i.e., $\sigma = \frac{card(A \bowtie B)}{card(A) \cdot card(B)}$. Table 1 summarizes these parameters. Table 2 contains some further notation.

**Appropriateness of our network model.** Note that our set of parameters describes each relation as a whole. There is an infinite number of concrete placements of nodes, which corresponds to each parameter setting.

PROPOSITION 3.1 *The results obtained for a parameter setting according to Table 1 apply to every instance of node placements that obeys the setting.*

This is true as we found that, given a specific parameter setting, the gain of sets of nodes $\{\mathcal{R}, \mathcal{A}_1, ..., \mathcal{A}_{n_A}, \mathcal{B}_1, ..., \mathcal{B}_{n_B}\}$ that obey this setting has a very small variance ($Var(gain) < 1\%$; cf. Section 4). Thus, every instantiation of our network model results in approximately the same gain. This makes our compact model very well suited for the analysis.

### 3.5 Cost Model

Our optimization goal is to minimize the energy consumed for communication. To this end, we need to model the communication costs.

**Costs of sending a packet via multiple hops.** The costs of sending a packet are computed by multiplying the one-hop costs with the number of hops. We model the one-hop costs as a parameter which is discussed at the end of this section ($c_a, c_b$). The number of hops is approximated by the Euclidean distance $d(\cdot, \cdot)$ between the sending and receiving node. Thus, the costs of sending a packet containing Tuple $t_A$ from node $\mathcal{A}_1$ to $\mathcal{A}_2$ are modeled as $d(\mathcal{A}_1, \mathcal{A}_2) \cdot c_a$.

Since we are interested in the relative performance of different strategies (gain), a model that provides costs proportional to the communication costs suffices. The Euclidean distance is proportional to the number of hops given the following assumption:

ASSUMPTION 3.2 *The sensor network is sufficiently dense such that the Euclidean distance between two nodes is (approximately) proportional to the number of hops.*

**Nodes at optimal locations.** In our analysis we will come up with strategies that perform computations at the mathematically optimal locations. Thus, our model assumes that there exists a node at the derived location. In practice, the node closest to this location has to be chosen. If the network is sufficiently dense, this node should be within communication distance of the optimal location. As a result, the number of hops will be the same or will differ by at most one hop (more or less). Therefore, we expect the influence on the results to be small.

ASSUMPTION 3.3 *The sensor network is sufficiently dense such that there exists a node within communication distance of each point.*

**Communication costs per hop:** In the remainder of this section we discuss the communication costs per hop ($c_a, c_b$). In addition to defining them, we discuss the range of these parameters for realistic communication hardware. This is important to derive meaningful conclusions.

The costs of sending one packet over one hop can be decomposed to *fixed costs* per packet and *variable costs* depending on the size of the payload: $cost(size(t)) = cost_{fix} + cost_{var}(size(t))$. We consider the costs of sending $t_A$, $t_B$ as well as of sending a result tuple $t_{AB}$. In order to reduce the number of parameters, we normalize the costs with respect to the costs of sending one result tuple: $c_a = \frac{cost(size(t_A))}{cost(size(t_{AB}))}$.

**Variable costs** depend on the size of the tuple. To interpret our results we assume:

ASSUMPTION 3.4 *The maximum size of a tuple is 15 attributes of two bytes.*

Note that 15 attributes is a lot given that current sensor nodes like Mica motes are equipped with up to 8 sensors. Thus, $t_{AB}$ can be up to 30 bytes larger than $t_A$ if $t_B$ is at maximum size, and no join attributes are projected out. In order to understand the lower bound on the size of $t_{AB}$, consider the number of join attributes. It is always possible to construct examples consisting of an arbitrary number of join attributes. However, in order to arrive at meaningful conclusions we focus on realistic scenarios:

ASSUMPTION 3.5 *The number of join attributes in sensor networks is at most 8.*

Thus, $t_{AB}$ can be up to 16 bytes smaller than $t_A$ if all 8 join attributes from $t_A$ and $t_B$ are projected out.

**Fixed costs** for sending one packet mainly depend on the MAC and PHY layer overhead. It results from the wakeup of the transmitter, carrier sensing, RTS and CTS, preamble, etc. In order to quantify these costs we looked at a sample of prominent MAC protocols: S-MAC [18], B-MAC [19], and SCP-MAC [20]. The minimum PHY/MAC layer overhead we observed is equivalent to the transmission of 127 bytes. This leads to the following assumption:

ASSUMPTION 3.6 *The difference in the energy consumption between sending an empty frame and a frame with 16 bytes payload is less than 15%. The difference for sending 30 bytes is less than 30%.*

This percentage is further reduced by overhearing, contention, errors and collisions. Most of the measurements we are aware of refer to the 802.11 protocol (e.g. [21]). There, increasing the payload by 30 bytes results in a difference of less than 10%. Consequently, the relative costs $c_a$ ($c_b$) are in the range from 0.75 (for a maximum of 30 bytes less than $t_{AB}$) up to 1.15 for 16 bytes more. This range should include any realistic communication hardware.

## 4 Centralized Join Processing

While our concern is distributed join processing, we start our analysis with the centralized case. We will state that all centralized join methods (e.g., [10, 12]) are optimal in specific scenarios only: The nodes involved need to be close to each other compared to their distance to the base station. In addition, a high selectivity is required.

This insight is interesting in its own right. [16] has arrived at similar findings by means of simulation. In contrast, given Proposition 3.1, our analytical approach rules out that there exist placements of nodes that are not in line with this result. Showing that centralized approaches are efficient in specific scenarios only motivates our examination of the distributed case. In addition, there are two reasons for presenting the analysis of the centralized case: The presentation of the distributed case becomes easier. Further, we will reduce parts of the analysis of the distributed case to the centralized one.

### 4.1 Cost Model for Centralized Strategies

Our goal is to identify scenarios where using a centralized processing at a single site $J$ results in energy savings compared to the root strategy, i.e., where there is a gain ($1 - \frac{cost_J}{cost_{root}} > 0$). In the following, we provide a model of the costs of centralized strategies based on our model of communication costs (cf. Section 3).

The cost of computing the join at (any) point $J$ is the sum of the costs of sending the tuples of Relations A and B to $J$ and sending the result to the root node subsequently:

$$cost_J = \sum_{i=1}^{n_A} c_a \cdot d(\mathcal{A}_i, J) + \sum_{i=1}^{n_B} c_b \cdot d(\mathcal{B}_i, J) + n_A \cdot n_B \cdot \sigma \cdot d(J, \mathcal{R})$$

$d(P_1, P_2)$ denotes the Euclidean distance. Recall that a root strategy is a centralized strategy, i.e.,

$$cost_{root} = \sum_{i=1}^{n_A} c_a \cdot d(\mathcal{A}_i, \mathcal{R}) + \sum_{i=1}^{n_B} c_b \cdot d(\mathcal{B}_i, \mathcal{R})$$

What remains to be specified for the model is the join location $J$. The optimal join location is not a parameter but depends on the placement of the nodes.

PROPOSITION 4.1 *It is impossible to derive a closed formula for the gain, irrespective of the parameterization of the network.*

PROOF. The join location that minimizes $cost_J$ is optimal. This corresponds to the Fermat problem [16]: For a given set of points $\{P_1, ..., P_n\}$ and their corresponding weights $\{w_1, ..., w_n\}$, find a point $J$ that minimizes $\sum_i w_i \cdot d(P_i, J)$. It has been shown that there is no closed expression for computing the Fermat point [22]. □

The Fermat problem can only be solved numerically.

### 4.2 Method

Proposition 4.1 results in two problems: (1) We need a method to compute the gain numerically. (2) We must be able to analyze the gain-function in order to identify global and local optima.

| | | |
|---|---|---|
| $n_A, n_B$ | Number of tuples of $A$, $B$ | 200, 300 |
| $l_{\mathcal{A}_\mathcal{C}\mathcal{R}}, l_{\mathcal{B}_\mathcal{C}\mathcal{R}}$ | Distance $\mathcal{A}_\mathcal{C}$, $\mathcal{B}_\mathcal{C}$ to $\mathcal{R}$ | 1.0, 1.0 |
| $\omega$ | Angle $\mathcal{A}_\mathcal{C}\mathcal{R}\mathcal{B}_\mathcal{C}$ | 0.5 (30°) |
| $\bar{d}_A, \bar{d}_B$ | Mean distance to $\mathcal{A}_\mathcal{C}$, $\mathcal{B}_\mathcal{C}$ | 0.5, 0.5 |
| $c_a, c_b$ | Relative costs of sending $t_A, t_B$ | 1.0, 1.0 |
| $\sigma$ | Selectivity | 0.002 |

**Table 3.** Standard setting for the analysis

**(1) Computation of the gain-function:** Our approach for numerically computing the Fermat point $F$ of a set of points ($\{\mathcal{R}, \mathcal{A}_1, ..., \mathcal{A}_m, \mathcal{B}_1, ..., \mathcal{B}_n\}$) is based on Weiszfeld's algorithm [23]. In order to provide strict confidence bounds, we compute the function gain($l_{\mathcal{A}_\mathcal{C}\mathcal{R}}, \bar{d}_A, n_A, c_a, l_{\mathcal{B}_\mathcal{C}\mathcal{R}}, \bar{d}_B, n_B, c_b, \omega, \sigma$) based on the Monte Carlo method as follows:

For a setting ($l_{\mathcal{A}_\mathcal{C}\mathcal{R}}, \bar{d}_A, n_A, c_a, l_{\mathcal{B}_\mathcal{C}\mathcal{R}}, \bar{d}_B, n_B, c_b, \omega, \sigma$) do:

1. Generate a random set of points $\{\mathcal{R}, \mathcal{A}_1, ..., \mathcal{A}_m, \mathcal{B}_1, ..., \mathcal{B}_n\}$ that follows the parameter setting.
2. Compute the Fermat point $F$
3. Compute the expected gain based on $cost_J$ with $J = F$

Aggregate the expected gain with the results from former trials and repeat until the confidence for the expected gain over all trials is within 0.01% with 98% probability.

According to our analysis, the variance of the gain is extremely small ($Var(gain) < 1\%$) for different sets of points that obey the same parameter setting. Thus, the expected gain is a reasonable measure to compare join strategies.

**(2) Analyzing the gain function.** We want to identify the optima of the gain function. Analytically finding optima requires differentiating the function. However, this is impossible for the gain due to Proposition 4.1. It is also problematic to find optima based on numerical methods: Such methods inspect a discrete number of values and make assumptions about the values in between. Thus, making reasonable assumptions is essential for ensuring not to miss local optima. For our analysis, we approach the problem twofold: In Section 4.3 we observe that the parameters are monotonic within the range defined in Subsection 3.5. In Section 4.4 we prove that our numerical approach finds the single global optimum for the gain. This proof is independent of our monotonicity assumptions. In addition, the proof further substantiates the monotonicity assumptions as they are in line with the global optimum.

### 4.3 Monotonicity Assumptions

This section deals with deriving the assumptions required for ruling out local optima of the gain-function. In particular, we derive monotonicity assumptions based on reasoning about the underlying problem. Note that the gain function is not differentiable and therefore it is impossible to prove its monotonicity. Thus, we first discuss the rationale behind our assumptions, and then complement our discussion by computing the values of the gain function at discrete points. Since the gain depends on 10 parameters, it is impossible to provide an exhaustive numerical scan over the full parameter space. We use

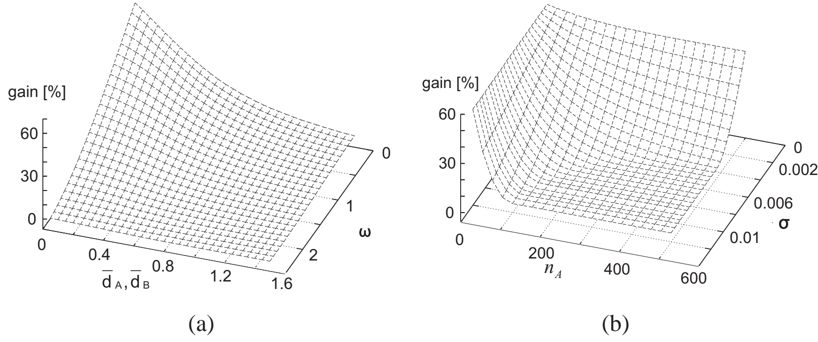


**Fig. 3.** (a) Influence of $\sphericalangle\mathcal{A}_c\mathcal{R}\mathcal{B}_c$ & distribution; (b) Influence of the result's size

a systematic approach similar to partial differentiation, i.e. we consider the parameters in isolation. In particular, we compute the gain numerically by systematically varying one of the parameters and set the other ones to a standard setting (Table 3). This setting is chosen such that it yields a medium gain (30%), i.e., it enables us to observe increases as well as decreases in the gain.

We now establish the monotonicity for the parameters related to the locations of the nodes, followed by the other parameters.

**Influence of the Tuple Locations.** We start discussing location-related parameters $(l_{\mathcal{A}_C\mathcal{R}}, \bar{d}_A, l_{\mathcal{B}_C\mathcal{R}}, \bar{d}_B, \omega)$ by providing an explanation based on characteristics of the Fermat point. Figure 2 serves as an illustration. The Fermat point is located "in between" the relations. If the angle or the mean distance of the nodes to their center becomes larger, the root node comes closer to also being "in between" the relations. In this case the root strategy and the centralized in-network strategy are similar, and the relative gain of the centralized in-network strategy becomes small.

ASSUMPTION 4.1 *The gain increases monotonically as the angle $\omega$ between the centers of the relations or the mean distance of the nodes to their centers ($\bar{d}_A$, $\bar{d}_B$) decreases. It decreases monotonically as the difference between $l_{\mathcal{A}_C\mathcal{R}}$ and $l_{\mathcal{B}_C\mathcal{R}}$ increases.*

Figure 3(a) illustrates the monotonicity for the mean distances of the nodes to their centers of mass ($\bar{d}_A$, $\bar{d}_B$) and the angle $\omega$. The discussion for $l_{\mathcal{A}_C\mathcal{R}}$ and $l_{\mathcal{B}_C\mathcal{R}}$ is analogous. Besides the monotonicity, the Figure 3(a) shows that the in-network strategy yields the maximum energy savings if all nodes except the root node are located close to each other ($\bar{d}_A = \bar{d}_B = 0$ and $\omega = 0$). This minimizes the routes along which the input tuples are sent.

**Influence of the Result Size.** Again, we start the discussion of the parameters related to the size of the result $(n_A, n_B, \sigma, c_a, c_b)$ by providing an explanation based on the Fermat point. It is known that as soon as the weight of one of the points is more than half of the total weight, it is the Fermat point [24]. Furthermore, if the weight of one of the points is close to half of the total weight, the Fermat point will be near that point. In our context, the weight of a node is the number of tuples that it sends multiplied with the transmission costs. More specifically, the weight of the result is $n_A \cdot n_B \cdot \sigma$, and the weights of the input tuples are $n_A \cdot c_a$ and $n_B \cdot c_b$. If $n_A \cdot n_B \cdot \sigma \geq m \cdot c_a + n \cdot c_b$,

the Fermat point will coincide with the root node. In this case, $cost_J$ and $cost_{root}$ are identical, and the gain will become $0$. In addition, the larger the size of the result, the closer will the Fermat point be to the root node.

ASSUMPTION 4.2 *Computing the result inside the network is only beneficial if the cardinality of the result is smaller than the input. In particular, this requires a high selectivity. Thus, the gain is monotonic in the parameters that determine the size of the result:* $c_a, c_b, n_A, n_B, \sigma$.

We complement the explanation by computing the gain function. Figure 3(b) shows the gain depending on $n_A$ (the number of nodes of Relation A) and the selectivity $\sigma$. The remaining parameters correspond to the standard setting. The figure confirms our assumption: As soon as the size of the result outweighs the input tuples, the gain becomes small, and any in-network strategy does not pay off. Furthermore, the gain is monotonic in the parameters that determine the size of the result.

### 4.4 Gain of Centralized Strategies

In the following, we present the single global optimum of the gain function:

PROPOSITION 4.2 *The gain of centralized in-network strategies has its maximum if the nodes are co-located (*$l_{\mathcal{A}_C\mathcal{R}} = l_{\mathcal{A}_C\mathcal{R}}$*,* $\bar{d}_B = \bar{d}_B = 0$*,* $\omega = 0$*) and the size of the result is minimal (0).*

PROOF. We have to show that the scenario in Proposition 4.2 is the global optimum. This can be seen by considering the range of the gain-function $[0, 1]$ (cf. Corollary 3.1). In the situation that we identified as a candidate for an optimum, $cost_J = 0$, while $cost_{root}$ takes on some fixed amount. In this case, the gain becomes 1 and thus is indeed a global optimum. Finally, this is the only global optimum. We conclude this from the formula (gain = $1 - \frac{cost_J}{cost_{root}}$) since $cost_J > 0$ (cf. Equation 4.1) for any other setting.□

Due to its monotonicity (cf. Assumptions 4.1 and 4.2) the gain-function has no local optima.

**Concluding remarks.** Centralized approaches can be more efficient than computing the join at the base station in rather specific scenarios only. To actually choose among the root strategy and a centralized in-network strategy, one would have to gather knowledge at the granularity of complete relations, more specifically: ($l_{\mathcal{A}_C\mathcal{R}}, \bar{d}_A, n_A, c_a, l_{\mathcal{B}_C\mathcal{R}}, \bar{d}_B, n_B, c_b, \omega, \sigma$). This is sufficient as the gain is insensitive to the concrete set of points as long as they obey the same parameter setting (cf. Proposition 3.1). Our objective in this paper is to find out if it is possible to design a join method which is efficient for general scenarios. As centralized approaches cannot achieve this, the question is whether distributing the processing results in more general join methods.

## 5 Distributed Join Processing

### 5.1 Logical Steps of Distributed Joins

To structure our analysis of distributed join strategies, we subdivide the processing into logical steps. As a motivation we briefly discuss two naive ideas for join distribution.

**Idea 1:** Process each tuple on the way from the sensor node to the root - one could simply form results tuples whenever two tuples meet that fulfill the join condition.

The problem is that we do not know whether the tuples have further join partners. In order to ensure correctness (Requirement 1, cf. Section 3) we would have to forward the input tuples along with the result tuples. As the root strategy only ships the input tuples, Idea 1 would be less efficient.

**Idea 2:** A more elaborate idea would be to route the tuples to join locations based on their join attributes (e.g., by hashing them onto join locations).

However, recall Requirement 2: A distributed approach makes sense only if we reduce the overall routing lengths compared to a centralized strategy. Sending tuples to arbitrary locations does not minimize the routes.

The important insight is that we *explicitly* need to ensure correctness and efficiency. Given these requirements, we structure distributed join processing as follows:

1. **Set up**:
   (a) Derive the knowledge necessary for guaranteeing correctness and efficiency.
   (b) Devise the optimal strategy, i.e., optimal join location(s) and the corresponding routes.
2. **Result computation**:
   (a) Send the tuples to its join location(s).
   (b) Compute the result.
   (c) Send the result to the base station.

Note that the purpose of these steps is to illustrate the problems and decisions involved in *optimally* distributing the processing. In particular, these are *logical* steps. There might be different ways of addressing them:

EXAMPLE 5.1 Consider the knowledge for ensuring that each pair of joining tuples meets at (at least) one join location (correctness). One way to obtain this knowledge is to derive from the query where potentially joining tuples are located. To illustrate, Yiu et al. [14] rely on the join condition: $distance(A, B) \leq d$ where $d$ is less than the communication range (cf. Section 2). An alternative way of guaranteeing correctness is by collecting the knowledge required explicitly. For instance, Yiu et al. [14] propose to introduce a pre-computation that identifies tuples that join. □

**Set up phase:** According to Section 4.4 the optimal central join location can be determined based on knowledge at the granularity of complete relations. In contrast, using knowledge at this granularity to set up an optimal distributed strategy is a problem: If one does not know which tuples join with each other, correctness (Requirement 1) requires sending one of the relations to every join location in its entirety. This should be the smaller one. The other relation is fragmented among the processing sites. If we considered just one of the processing sites, the discussion from Section 4 applies. Thus, a distributed strategy based on relation-level knowledge would suffer from the same problems as centralized strategies: It would be efficient only for very specific scenarios. As we are interested in distributed strategies that are more general than the centralized strategy, our analysis concentrates on knowledge at the granularity of nodes.

It is an open question which knowledge (at the granularity of nodes) is required to devise an *optimal* distribution. Do we need to know the join partners of a tuple? Do we

need to know their locations to minimize routes? To address this problem, we start our analysis by assuming complete knowledge. In particular, we exactly know which tuples join with each other and where matching tuples are located. We relax this assumption at the end of our analysis.

### 5.2 Using Knowledge at Node Granularity

Knowledge at the granularity of nodes means that we can infer which tuples have no join partner. Since an optimal strategy would not send out these tuples, knowledge at node granularity leads to a *distributed filtering*. Distributed filtering is applicable to distributed strategies as well as to the root strategy:

DEFINITION 5.1 (Root Strategy with Filtering) *A root strategy with filtering discards tuples that do not join and then only sends the remaining tuples to the base station.*

Even though the root strategy with filtering computes the result at a single node, it can be seen as distributed join processing as the set up phase is distributed. Note that filtering cannot be combined with a centralized in-network strategy: According to Section 4 the latter requires a high selectivity. This is not fulfilled after filtering out tuples that do not join. We conclude:

COROLLARY 5.1 *Knowing which tuples do not join can be exploited for distributed filtering. This is orthogonal to distributing the result computation, i.e., filtering can also be combined with the root strategy. It cannot be combined with a centralized in-network strategy.* □

Figure 1 serves as an illustration of Corollary 5.1. Finally, we briefly discuss the gain when combining the filtering with the root strategy.

PROPOSITION 5.1 *Filtering requires knowing which tuples do not join. Given this knowledge, filtering leads to a gain depending on the fraction of tuples that join.*

PROOF. In the result-computation step the root strategy with filtering sends only the *contributing* tuples, i.e., the tuples that join. Leaving aside the costs of obtaining the knowledge, the root strategy with filtering can save up to 100% if none of the tuples join. At the other extreme, if every tuple contributes to the result, the root strategy with filtering is as costly as the root strategy (0% gain). □

The proposition indicates that efficiently collecting this knowledge is a promising direction for join processing. Note that the filtering exploits only a fraction of the complete knowledge that we are currently assuming.

### 5.3 Devising an Optimal Distributed Strategy

Our interest is in analyzing whether we can achieve a gain by optimally distributing the result computation, i.e., Steps 2 a, b, c. In the following, we focus on an *optimal* distributed strategy. It upper bounds the gain of distributed strategies. In particular, an optimal strategy avoids to send tuples that do not join. Therefore, the subsequent discussion is restricted to joining tuples.

The major difficulty in analyzing how to optimally distribute the result computation is that optimal locations and optimal routes mutually depend on each other. We start

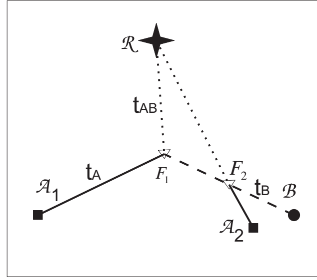


**Fig. 4.** Example of two join locations

the discussion with a concept that identifies subsets of tuples that can be regarded in isolation:

DEFINITION 5.2 (Group) *A group of tuples $G$ is a subset of the union of Relations A and B such that if tuple $t \in G$ then every tuple $t'$ that joins with $t$ is in $G$ as well.*

EXAMPLE 5.2 For an equi-join a group is a subset of Relations A and B that yields a cross product. In this case, if we restrict the join to a single group its selectivity $\sigma = 1$. □

The following corollary is a consequence of Definition 5.2:

COROLLARY 5.2 *The processing locations of different groups are independent of each other.* □

The reason is that tuples from different groups do not have to be brought together at one location. This property lets us restrict our analysis to a single group in order to find the optimal distribution. We start with a simple example of a distributed processing that we will use in subsequent discussions:

EXAMPLE 5.3 Figure 4 shows two nodes $\mathcal{A}_1, \mathcal{A}_2$ belonging to Relation A and one node $\mathcal{B}$ of Relation B. Assume that their tuples form a group and all tuples have the same size. The figure shows a strategy where $t_B$ is first joined with the tuple from $\mathcal{A}_2$. The result is sent to the root node. $t_B$ is also sent to a second location where it is joined with the tuple from $\mathcal{A}_1$. □

Our analysis how to distribute the result computation is structured as follows:

1. For a group of tuples: identify the optimal number of processing sites
2. For a group of tuples: analyze where these sites are located

**Optimal number of processing sites** The difficulty in identifying the optimal number of sites is that our problem cannot be reduced to a solved mathematical one. However, we can upper bound the gain by identifying the scenario with the highest gain when distributing the computation of the join result:

DEFINITION 5.3 ($n_A$:1 Scenario) *An $n_A$:1 scenario is a group consisting of $n_A$ tuples of Relation A that pair up with one tuple of Relation B.*

PROPOSITION 5.2 *The $n_A$:1 scenario is the optimal case for multiple join locations, compared to all other groups ($n_A : n_B$), for a fixed set of nodes from Relation $A$.*

PROOF. If $n_A, n_B \geq 2$ (otherwise we have an $n_A$:1 scenario), the optimal centralized join location is the root, as the cardinality of the result is larger than the cardinality of the input (property of the Fermat point [24]). Consider increasing $n_B$ by 1. If the computation is centralized, this means that we have to send $t_B$ to the root. In the distributed case, we have to send $t_B$ to each of the processing locations. Afterwards, $n_A$ result tuples have to be sent from the processing sites to the base station. Thus, the cost increase for the centralized setting is less in relative terms. Therefore, the $n_A$:1 scenario yields the largest gain for a distributed computation, compared to a centralized one. □

In the $n_A$:1 scenario, the tuples can be joined at many locations and in different orders, resulting in a combinatorial problem. We devise a method for upper bounding the gain that consists of:

- an algorithm for computing optimal strategies for two or three tuples in Relation A ($n_A \in \{2, 3\}$), and
- an estimator for lower bounding the costs for $n_A > 3$.

**Computing Optimal Strategies.** We can compute optimal strategies based on the following proposition:

PROPOSITION 5.3 *Any join location $J$ in a distributed strategy is a Fermat point. It minimizes the routes of the nodes from which a tuple is sent to $J$ and to which a tuple is sent from $J$.*

PROOF. The proof is the same as the proof of the corresponding property of Steiner trees [25]. The idea is that the routes could be further optimized if the join location was not the Fermat point. □

For the case $n_A = 2$, Proposition 5.3 restricts the number of possible strategies to three:

1. Join $\mathcal{A}_2$ with $\mathcal{B}$ at Fermat Point $\mathcal{F}_2$ and send $t_B$ on to $\mathcal{F}_1$ and join it with the tuple from $\mathcal{A}_1$ (cf. Figure 4).
2. Join $\mathcal{A}_1$ with $\mathcal{B}$, then join $\mathcal{A}_2$ with $\mathcal{B}$.
3. Use one Fermat point for all nodes $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{B}$ and $\mathcal{R}$.

By computing the join locations in all three cases and comparing the overall costs we find the optimal strategy. For $m = 3$ we can follow the same procedure except that there are 13 possible constellations.

**Lower bound estimation.** For $n_A > 3$ we lower bound the costs (upper bound the gain) of processing the join by reducing the problem to $n_A = 3$. We accomplish this reduction by choosing two distinguished nodes from Relation A and assume that the rest was located at one third point. We choose the two distinguished nodes by taking the node closest to the root as the first point and the node $N$ that maximizes $d(N, \mathcal{R}) + d(N, \mathcal{B})$ as the second. The intuition is to take the distribution of the nodes into account in order to arrive at a meaningful estimation of the communication costs. The remaining nodes of Relation A are assumed to be located at a single point on the line $\mathcal{R}\mathcal{A}_\mathcal{C}$. In this way, we keep the mean distance from the root node unchanged. Assuming the remaining nodes to be co-located leads to an underestimation of the costs as it reduces the routing lengths to pair the tuples. Figure 2 serves as an illustration of the third point.

**Gain of Several Join Locations.** In the following we compute a lower bound of the costs of computing the join and compare it to the optimal centralized strategy. Again,

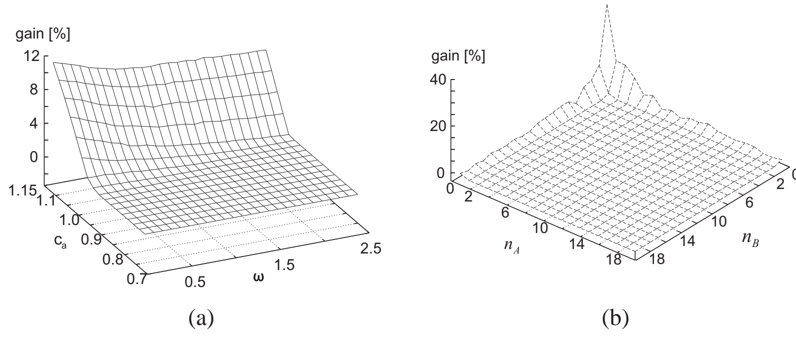


**Fig. 5.** (a) Gain: optimal number vs. single site; (b) Gain: optimal location vs. root node

the following discussion is based on monotonicity assumptions of the gain. This is analogous to Section 4.

PROPOSITION 5.4 *Using multiple join locations per group of pairing tuples results in energy savings of at most 12% as compared to choosing a single site. This is an upper bound on the savings for the optimal scenario ($n_A$:1) and holds if the tuples of Relation A are larger than the result tuples.*

We found the costs $c_a$ of sending a tuple of Relation A to be the most influential parameter. Figure 5(a) shows its influence along with the angle $\omega$ between Relation A and B. Intuitively, if the result tuples are larger than the ones of Relation A ($c_a < 1$), sending the input tuples directly to the root node is a good choice. Thus, the optimal centralized as well as the optimal decentralized strategy are alike. Only if the result tuples are much smaller than the ones of Relation A, multiple join locations can reduce the energy consumption. Our analysis confirms this intuition. If $c_a < 1$ using a single join location is optimal. In addition, Figure 5(a) already shows the maximum gain: In analogy to Section 4, the gain computed confirms the monotonicity assumptions. Therefore, we have identified the globally optimal gain.

Our analysis of the scenario with the maximum gain reveals that the upper bound for the energy savings of multiple locations per group of joining tuples is small. Thus, the number of join locations should be one per group.

**Location of Optimal Site per Group** In order to identify the single optimal join location per group, we compare computing the result for a group at its Fermat point to computing it at the root.

PROPOSITION 5.5 *In settings with a filtering the optimal location per group is the root node if $n_A, n_B \geq 2$; the location is the same for every group.*

The analysis for one group is the same as the one in Section 4. The only difference is that the selectivity factor is high (selectivity is low), at least $\frac{n_A}{n_A+1}$ for the $n_A$:1 scenario, and the number of tuples per group might be small. Thus, the results directly apply. The most important influence is the size of the result, which is larger than the input if $n_A, n_B \geq 2$. Figure 5(b) visualizes this influence, depicting the percental gain of the optimal location. As soon as the group consists of more than one tuple per relation ($n_A, n_B \geq 2$) the root node is the optimal location. Only if $n_A = n_B = 1$ a distributed

join strategy can save up to 50% energy if two input tuples lead to one result tuple. Note that $n_A = n_B = 1$ means that none of the two joining tuples has a further join partner.

**Conclusion.** Our most important insight is that the result computation is optimally performed at the root node, if all tuples that do not join are filtered in advance. Joining tuples at multiple locations does not increase the energy-efficiency. This result applies for every join method that involves a filtering. A distributed strategy which is more general than the centralized strategy but does not involve a filtering is impossible because devising a distribution requires knowledge at the granularity of single nodes. Finally, the distributed filtering is a promising direction to process join queries efficiently if only a small fraction of the tuples join, and the knowledge can be acquired efficiently. This in turn settles the knowledge at the granularity of nodes that is required: We need to know which tuples join. Most notably, information beyond that is not helpful to optimize the processing further.

## 6 Conclusions

In sensor networks, simple query operators are optimally executed by reducing the amount of data close to the sources. This requires a distributed processing. In contrast, distributing the computation of joins is an open problem. In this paper, we aimed at theoretical insights in how to efficiently distribute the join. We were interested in analytically identifying parts of the problem for which we can derive optimal solutions. Our most important contribution is to show that joining tuples at multiple locations does not increase energy-efficiency. This result applies for every join method that involves a filtering. We showed that after the filtering the result is optimally computed at the base station. As devising an optimal strategy only makes sense based on knowledge at the granularity of single nodes, the filtering is inherently contained in a distributed processing. At the same time, the filtering can result in substantial savings if only a small fraction of tuples joins. Thus, developing efficient methods for finding out which of the tuples join and subsequently joining them at the base station is the most promising direction towards an efficient join processing and is in focus of our ongoing work.

***Acknowledgments.*** *This work was partially supported by the German Research Foundation (DFG) within the Research Training Group GRK 1194 "Self-organizing Sensor-Actuator Networks" (GRK1194).*

## References

1. Abadi, D.J., Madden, S.R., Lindner, W.: REED: Robust, Efficient Filtering and Event Detection in Sensor Networks. In: Proceedings of the 31st International Conference on Very Large Data Bases (VLDB'05), VLDB Endowment (2005) 769–780
2. Yao, Y., Gehrke, J.: Query processing for sensor networks. In: Proceedings of the 1st Biennial Conference on Innovative Data Systems Research (CIDR'03). (January 2003)
3. Madden, S., Franklin, M.J., Hellerstein, J.M., Hong, W.: The design of an acquisitional query processor for sensor networks. In: SIGMOD '03, ACM Press (2003) 491–502
4. Madden, S.R., Franklin, M.J., Hellerstein, J.M., Hong, W.: Tinydb: an acquisitional query processing system for sensor networks. ACM Transactions on Database Systems (TODS) **30**(1) (2005) 122–173

5. Yao, Y., Gehrke, J.: The cougar approach to in-network query processing in sensor networks. ACM SIGMOD Record **31**(3) (2002) 9–18
6. Gehrke, J., Madden, S.R.: Query processing in sensor networks. IEEE Pervasive Computing **03**(1) (2004) 46–55
7. Nath, S., Gibbons, P.B., Seshan, S., Anderson, Z.R.: Synopsis diffusion for robust aggregation in sensor networks. In: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys'04). (2004)
8. Considine, J., Li, F., Kollios, G., Byers, J.: Approximate aggregation techniques for sensor databases. In: ICDE'04, Washington, DC, USA (2004) 449
9. Bonfils, B.J., Bonnet, P.: Adaptive and decentralized operator placement for in-network query processing. Telecommunication Systems **26** (June 2004) 389–409
10. Chowdhary, V., Gupta, H.: Communication-Efficient Implementation of Join in Sensor Networks. In: Proceedings of the 10th International Conference on Database Systems for Advanced Applications (DASFAA'05). Volume 3453., Springer (2005) 447–460
11. Pandit, A., Gupta, H.: Communication-Efficient Implementation of Range-Joins in Sensor Networks. In: Proceedings of the 11th International Conference on Database Systems for Advanced Applications (DASFAA'06). (2006) 859–869
12. Coman, A., Nascimento, M.A.: A distributed algorithm for joins in sensor networks. In: Proceedings of the 19th International Conference on Scientific and Statistical Database Management (SSDBM '07), Washington, DC, USA (2007)
13. Yu, H., Lim, E.P., Zhang, J.: On in-network synopsis join processing for sensor networks. In: Proceedings of the 7th International Conference on Mobile Data Management (MDM'06), Nara, Japan (2006)
14. Man Lung Yiu, Nikos Mamoulis, S.B.: Evaluation of spatial pattern queries in sensor networks. Technical Report HKU CS Tech Report TR-2007-02, University of Hong Kong
15. Yang, X., Lim, H.B., Özsu, T.M., Tan, K.L.: In-network execution of monitoring queries in sensor networks. In: SIGMOD '07. (2007) 521–532
16. Coman, A., Nascimento, M.A., Sander, J.: On join location in sensor networks. In: Proceedings of the 8th International Conference on Mobile Data Management (MDM'07), Mannheim, Germany (2007)
17. Madden, S., Franklin, M.J., Hellerstein, J.M., Hong, W.: TAG: A Tiny AGgregation service for ad-hoc sensor networks. In: Proceedings of the 5th Symposium on Operating System Design and Implementation (OSDI 2002), Boston, MA (December 2002)
18. Ye, W., Heidemann, J., Estrin, D.: Medium access control with coordinated adaptive sleeping for wireless sensor networks. IEEE/ACM Transactions on Networking **12**(3) (2004) 493–506
19. Polastre, J., Hill, J., Culler, D.E.: Versatile low power media access for wireless sensor networks. In: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys'04), Baltimore, MD, USA (2004) 95–107
20. Ye, W., Silva, F., Heidemann, J.: Ultra-low duty cycle mac with scheduled channel polling. In: Proceedings of the 4th international conference on Embedded networked sensor systems (SenSys'06), Boulder, Colorado, USA (2006) 321–334
21. Feeney, L.M.: An energy consumption model for performance analysis of routing protocols for mobile ad hoc networks. Mobile Networks and Applications **6**(3) (2001) 239–249
22. Bajaj, C.: The algebraic degree of geometric optimization problems. Discrete Comput. Geom. **3**(2) (1988) 177–191
23. Kuhn, H.W.: Steiner's problem revisited. In: G. B. Dantzig and B. C. Eaves, editors, Studies in Optimization. Studies in Mathematics, 10, The Mathematical Association of America, Washington, DC (1974) 52–70
24. Tollisen, G.P., Lengyel, T.: On minimizing distance by the road less travelled. Elem. Math. **58**(3) (2003) 89–107
25. Melzak, Z.A.: On the problem of steiner. Canad. Math. Bull. **4**(2) (1961) 143–148