Cost-Optimal Execution of
Boolean Query Trees with Shared Streams

Henri Casanova[1], Lipyeow Lim[1], Yves Robert[2,3], Frédéric Vivien[2], and Dounia Zaidouni[2]

1. *University of Hawai'i at Manoa, Honolulu, USA*
{henric|lipyeow}@hawaii.edu
2. *École Normale Supérieure de Lyon & INRIA, France*
{Yves.Robert|Frederic.Vivien|Dounia.Zaidouni}@ens-lyon.fr
3. *University of Tennessee Knoxville, USA*

Abstract—The processing of queries expressed as trees of boolean operators applied to predicates on sensor data streams has several applications in mobile computing. Sensor data must be retrieved from the sensors, which incurs a cost, e.g., an energy expense that depletes the battery of a mobile query processing device. The objective is to determine the order in which predicates should be evaluated so as to shortcut part of the query evaluation and minimize the expected cost. This problem has been studied assuming that each data stream occurs at a single predicate. In this work we remove this assumption since it does not necessarily hold in practice. Our main results are an optimal algorithm for single-level trees and a proof of NP-completeness for DNF trees. For DNF trees, however, we show that there is an optimal predicate evaluation order that corresponds to a depth-first traversal. This result provides inspiration for a class of heuristics. We show that one of these heuristics largely outperforms other sensible heuristics, including a heuristic proposed in previous work.

I. INTRODUCTION

There has been a recent explosion in the use of personal mobile devices for "mobile sensing" applications. For instance, smartphones are equipped with increasingly sophisticated sensors (e.g., GPS, accelerometer, gyroscope, microphone) that enable near real-time sensing of an individual's activity or environmental context. A smartphone can then perform embedded query processing on the sensor data streams, e.g., for social networking [1], remote health monitoring [2]. The continuous processing of streams, even when data rates are moderate (such as for GPS or accelerometer data), can cause commercial smartphone batteries to be depleted in a few hours [3]. It is thus crucial to reduce the amount of sensor data acquired for query processing, so as to reduce energy consumption and lengthen battery life.

In this work we study the problem of minimizing the expected sensor data acquisition cost (e.g., number of bytes, energy consumption due to byte transfers) when evaluating a query expressed as a tree of conjunctive and disjunctive boolean operators applied to boolean predicates. Each predicate is computed over data items from a particular data stream generated periodically by a sensor, and as a certain probability of evaluating to true. The evaluation of the query stops as soon as a truth value has been determined, possibly *shortcircuiting* part of the query tree. A "push" model by which sensors continuously transmit data to the device maximizes the amount of acquired data and is thus not practical. Instead, a "pull" model has been proposed [4], by which the query engine running on the device carefully chooses the *order* and the *numbers of data items* to request from each individual sensor. This choice is based on a-priori knowledge of operator costs and probabilities, which can be inferred based on historical traces obtained for previous query executions. Such intelligent processing is possible thanks to the programming and data filtering capabilities that are emerging on many wearable sensor platforms (e.g., the SHIMMER platform [5]) so that data storage and transmission algorithms can be programmed "over the air."

Two example query trees are shown in Figure 1, assuming streams named A, B, and C, which are assumed to produce integer data items. Each leaf corresponds to a boolean predicate. A predicate may involve no operator, e.g., "$C < 3$" is true if the last item from stream C is strictly lower than 3, or based on an arbitrary operator (in this example MAX or AVG) which is applied to a time-window for a stream, e.g., "$\mathsf{AVG}(A, 5) < 70$" is true if the average of the last 5 items from A is strictly lower than 70).

The problem of computing the truth value of a boolean query tree while incurring the minimum cost is known as Probabilistic AND-OR Tree Resolution (PAOTR) and has been studied extensively in the literature. In particular, [6] provides both a survey of known theoretical results and several new results, all assuming that each data stream occurs in at most one leaf of the query tree. This assumption is termed *read-once* therein. In this case, for AND-trees (i.e., single-level trees with an AND operator at the root node) a simple $O(n \log n)$ greedy
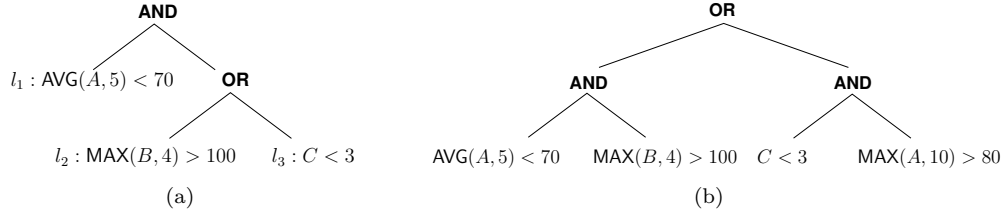
Figure 1. Two query tree examples: (a) a *read-once* query; (b) a *shared* query.

algorithm produces an optimal leaf evaluation order (n is the number of leaves in the query tree) [7]. For DNF trees (i.e., collections of AND-trees whose roots are the children of a single OR node), a $O(n \log n)$ depth-first traversal of the trees that reuses the algorithm in [7] to order leaves within each AND produces an optimal evaluation order [6]. For general AND-OR-trees the complexity of the problem is open. The example query tree in Figure 1(a) is a *read-once* query since no stream occurs in two leaves.

By contrast, in this work we study the more general case, which we term *shared*, in which a stream can occur in multiple leaves. The example in Figure 1(b) corresponds to a *shared* case since stream A occurs in two leaves. Such queries are fairly common in, among others, telehealth scenarios. For example, an alert may be generated either if the heart rate is high (e.g., > 100) and the accelerometer is stationary or if the heart rate is low and SPO2 (blood oxygen saturation) is low. It is thus not difficult to imagine sensor streams appearing in multiple leaf predicates. The device that processes the query acquires data items from streams and holds each data item in memory until that data item is no longer relevant. A data item from a stream is no longer relevant when it is older than the maximum time-window used for that stream in the query. Each time a leaf of the query must be evaluated, one can then compute the number of data items that must be retrieved from the relevant stream given the time-windows of the operator applied to that stream and the data items from that stream that are already in the device's memory. For example, considering the query in Figure 1(b), assume the predicate "$\mathsf{AVG}(A, 5) < 70$" is evaluated first, thus pulling 5 items from stream A. If later the predicate "$\mathsf{MAX}(A, 10) > 80$" needs to be evaluated then only 5 additional items must be pulled.

The *shared* scenario is important in practice, and has been introduced and investigated in [4]. In that work the authors do not give theoretical results, but instead develop heuristics to determine an order of operator evaluation that hopefully leads to low data acquisition costs. To the best of our knowledge, the complexity of the PAOTR problem in the *shared* case has never been addressed in the literature, likely because re-using

stream data across leaves dramatically complicates the problem. When picking a leaf evaluation order, inter-dependences between the leaves must be taken into account. In fact, even when given a leaf evaluation order, computing the expected query cost is intricate while this same computation is trivial in the *read-once* case.

In this work we study the PAOTR problem in the *shared* case and make the following contributions:

- For AND-trees we give an optimal algorithm (which is much more involved than the optimal algorithm in the *read-once* case);
- For DNF trees we show that the problem is NP-complete; but we are able to prove that there exists an optimal leaf evaluation order that is depth-first;
- For DNF trees we develop heuristics that we evaluate in simulation and compare to the optimal solution (computed via an exhaustive search) and to the heuristic proposed in [4].

In Section II we discuss models, the problem statement, and related work. We study AND-trees and DNF trees in Section III and Section IV, respectively. Section V concludes the paper with a brief summary of our findings and perspectives on future work. Detailed proofs of some of our theoretical results are provided in a technical report [8].

II. Problem Statement and Examples

To define our problem we reuse the formalism and terminology in [6]. A query is an AND-OR *tree*, i.e., a rooted tree whose non-leaf nodes are AND or OR operators, and whose leaf nodes are labeled with probabilistic boolean predicates. Each predicate is evaluated over data items generated by a data *stream*. The evaluation of each predicate has a known *success probability* (the probability that the predicate evaluates to TRUE) and a *cost*. In practice, the success probability can be estimated based on historical traces obtained from previous query evaluations. As in [6], we assume *independent* predicates, meaning that two predicates at two leaf nodes in a query are statistically independent. The cost is determined by the number of data items required to perform the evaluation and the evaluation cost per data item for the stream. For instance, the cost of a data item could correspond to the energy cost, in joules, of

acquiring one data item based on the communication medium used for the stream and the data item size.

More formally, we consider a set of s streams, $\mathcal{S} = \{S_1, \ldots, S_s\}$. Stream S_k has a cost per data item of $c(S_k)$. A query on these streams, $\mathcal{T}$, is a rooted AND-OR tree with m leaves, $l_1, \ldots, l_m$. Leaf l_j has success, resp. failure, probability p_j, resp. $q_j = 1 - p_j$, and requires the last d_j items from stream $S(j) \in \mathcal{S}$. The objective is to compute the truth value of the root of the query tree by evaluating the leaves of the tree. Because each non-leaf node in a query tree is either an OR or an AND operator, it may not be necessary to evaluate all the leaves due to *shortcircuiting*. In other words, as soon as any child node of an OR, resp. AND, operator evaluates to TRUE, resp. FALSE, the truth value of the operator is known and can be propagated toward the root. For a given query, we define a *schedule* as an evaluation order of the leaves of the query tree, represented as a sorted sequence of the leaves.

We define the *cost* of a schedule as the **expected value** of the sum of the costs incurred for all leaves that are evaluated before the root's truth value is determined. For instance, consider the query in Figure 1(a), in which leaves are labeled l_1, l_2, l_3, and consider the schedule l_2, l_3, l_1. The query processing begins with the acquisition of the data items necessary for evaluating l_2, which has cost $4 \cdot c(B)$. With probability p_2, l_2 evaluates to TRUE, thus shortcircuiting the evaluation of l_3. Therefore, the expected evaluation cost of the OR operator is: $4 \cdot c(B) + q_2 \cdot c(C)$. If the OR operator evaluates to FALSE, which happens with probability $q_2 q_3$, then the evaluation of l_1 is shortcircuited. Otherwise, l_1 must be evaluated. The overall cost of the schedule is thus: $4 \cdot c(B) + q_2 \cdot c(C) + (1 - q_2 q_3) \cdot 5 \cdot c(A)$. Recall that this query tree is for a *read-once* scenario.

The PAOTR problem consists in determining a schedule with minimum cost. The complexity of this problem is unknown in the *read-once* case for general AND-OR trees, while optimal polynomial-time algorithms are known for AND-trees [7] and DNF trees [6]. In this work, we focus on these two types of trees in the *shared* case, seeking to develop optimal algorithms or to show NP-completeness. We refer the reader to [6] for a detailed review of the PAOTR literature. To the best of our knowledge, the only work that has studied the *shared* case is [4], in which a heuristic is proposed for DNF trees. We evaluate this heuristic in Section IV-D. In the next two sections we give examples of cost computations for an AND-tree and a DNF tree both in the *shared* case.

A. AND-*tree example*

Consider the AND-tree query depicted in Figure 2 with three leaves labeled l_1, l_2, and l_3, for two streams A

and B. For each leaf (l_i), we indicate the stream ($S(i)$), the number of data items needed from that stream to evaluate the leaf (d_i), and the success probability (p_i). For instance, leaf l_2 requires $d_2 = 2$ items from stream $S(2) = A$ and evaluates to TRUE with probability $p_2 = 0.1$. We assume that retrieving a data item from a stream has unitary cost, regardless of the stream. There are 6 possible schedules for this tree, each schedule corresponding to one of the 3! orderings of the leaves. The optimal algorithm for *read-once* AND-trees sorts the leaves by non-decreasing $d_j c(S(j))/q_j$ [7]. Because $\frac{1 \times c(A)}{q_1} = \frac{1}{1-0.75} = 4$, $\frac{2 \times c(A)}{q_2} = \frac{2}{1-0.1} \approx 2.22$, and $\frac{1 \times c(B)}{q_3} = \frac{1}{1-0.5} = 2$, this algorithm schedules leaf l_3 first. There are two possible schedules with l_3 as the first leaf:

- l_3, l_1, l_2 whose cost is: $c(B) + p_3 \times (c(A) + p_1 \times c(A)) = 1 + 0.5 \times (1 + 0.75 \times 1) = 1.875$; and
- l_3, l_2, l_1 whose cost is: $c(B) + p_3 \times (2 \times c(A) + p_2 \times 0 \times c(A)) = 1 + 0.5 \times (2 + 0.1 \times 0) = 2$.

However, another schedule, l_1, l_2, l_3, has a lower cost: $c(A) + p_1 \times (c(A) + p_2 \times c(B)) = 1 + 0.75 \times (1 + 0.1 \times 1) = 1.825$. Therefore, the optimal algorithm for the PAOTR problem for *read-once* AND-trees is no longer optimal in the *shared* case.

B. DNF *tree example*

Figure 3 shows a DNF tree with three AND nodes, for four streams A, B, C, and D. Each leaf requires only one data item from a stream. Leaves are labeled l_1 to l_7, in the order in which they appear in a given schedule. This example is meant to illustrate the difficulty of the PAOTR problem in the case of DNF trees in the *shared* scenario. In particular, computing the cost of a schedule is much more complicated than in the *read-once* scenario due to inter-leaf dependencies. Let $\mathcal{C}_j$ be the cost of evaluating leaf l_j, and $\mathcal{C}$ the overall cost of the schedule. We consider the 7 leaves one by one, in order:

Leaf l_1 − The first leaf is evaluated: $\mathcal{C}_1 = c(A)$.

Leaf l_2 − This is the first leaf in its AND, no AND has been fully evaluated so far, and l_2 is the first encountered leaf that requires stream B. Therefore, l_2 is always evaluated, requiring a data item from stream B: $\mathcal{C}_2 = c(B)$.

Leaf l_3 − This is the second leaf from its AND, no AND has been fully evaluated so far, and l_3 is the first
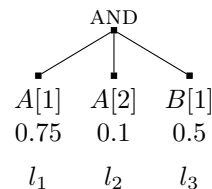

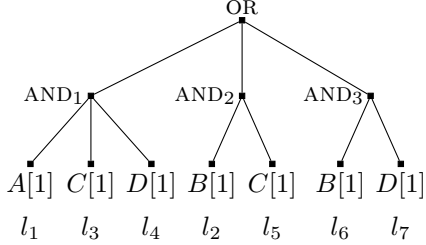Figure 2.   Example *shared* AND-tree.

Figure 3. Example DNF tree.

encountered leaf that requires stream C. Therefore, a data item from C is acquired if and only if l_1 evaluates to TRUE: $\mathcal{C}_3 = p_1 c(C)$.

Leaf l_4 – This is the third leaf from its AND, no AND has been fully evaluated so far, and l_4 is the first encountered leaf that requires stream D. Therefore, one data item is acquired from D if and only if l_1 and l_3 both evaluate to TRUE: $\mathcal{C}_4 = p_1 p_3 c(D)$.

Leaf l_5 – This is the second leaf from its AND, and AND$_1$ has been fully evaluated so far. However, one of the leaves of that AND, l_3, requires a data item that is also needed by l_5, from stream C. If l_3 has been evaluated, then the evaluation cost of l_5 is 0 because the necessary data item from C has already been acquired and is available "for free" when evaluating l_5. If l_3 has not been evaluated (with probability $1 - p_1$), it means that AND$_1$ has evaluated to FALSE. Then, if l_2 has evaluated to TRUE, l_5 must be evaluated thus requiring the data item from stream C. We obtain $\mathcal{C}_5 = (1 - p_1)p_2 c(C)$.

Leaf l_6 – Since l_2 is always evaluated the data item from stream B required by l_6 is always available for free: $C_6 = 0$.

Leaf l_7 – This is the second leaf from its AND, and AND$_1$ and AND$_2$ have been fully evaluated so far. However, one of the leaves of AND$_1$, l_4, but none of those of AND$_2$, require the data item that is needed by l_7 from stream D. Therefore, l_7 must be evaluated and its evaluation is not free if and only if l_4 has not been evaluated, AND$_2$ has evaluated to FALSE, and the evaluation of AND$_3$ went as far as l_7. Therefore, $\mathcal{C}_7 = (1 - p_1 p_3)(1 - p_2 p_5)p_6 c(D)$.

Overall, we obtain the cost of the schedule:

$$
\begin{aligned}
\mathcal{TC} &= c(A) + c(B) + (p_1 + (1 - p_1)p_2)c(C) \\
&+ (p_1 p_3 + (1 - p_1 p_3)(1 - p_2 p_5)p_6)c(D)
\end{aligned}
$$

Given the complexity of the above cost computation, one might expect the PAOTR problem to be NP-complete in the *shared* case (recall that it is polynomial in the *read-once* case). We confirm this expectation in Section IV.

III. AND TREES

In this section, we focus on AND-trees. We have seen in Section II-A that the simple greedy algorithm proposed in [7] in the *read-once* case is not optimal in the *shared* case. We propose an algorithm and we prove that it is optimal. This algorithm is still greedy but compares the ratios of cost to failure probability of all sequences of leaves that use the same stream, instead of only considering pair-wise leaf comparisons. We begin in Section III-A with a preliminary result on the optimal ordering of leaves that use the same stream.

A. Ordering same-stream leaves

In the example given in Section II-A, we considered two schedules that begin with leaf l_3. In the first schedule leaf l_1 precedes l_2, while the converse is true in the second schedule. Leaf l_1 requires one data item from stream A, while leaf l_2 requires two data items from the same stream. Therefore the first schedule is always preferable to the second schedule: if we evaluate l_1 before l_2 and if l_1 evaluates to FALSE, then there is no need to retrieve the second data item and the cost is lowered. A general result can be obtained:

Proposition 1. *Consider an* AND*-tree and a leaf l_i that requires d_i data items from a stream S. In an optimal schedule l_i is scheduled before any leaf l_j that requires $d_j > d_i$ data items from stream S.*

Proof: This proposition is proven via a simple exchange argument [8]. ∎

B. Optimal schedule

Consider an AND-tree with m leaves, $l_1, \ldots, l_m$, for s streams, $S_1, \ldots, S_s$. We define $\mathcal{L}_k = \{l_j | S(l_j) = S_k\}$, i.e., the set of leaves that require data items from stream S_k. Algorithm 1 shows a greedy algorithm (implemented recursively for clarity of presentation) that takes as input the $\mathcal{L}_k$ sets, an initially empty schedule ξ, and an array of s integers, *NItems*, whose elements are all initially set to zero. This array is used to keep track, for each stream, of how many data items from that stream have been retrieved in the schedule so far. Each call to the algorithm appends to the schedule a sequence of leaves that require data items from the same stream, in increasing order of number of data items required. The algorithm stops when all leaves have been scheduled. The algorithm first loops through all the streams (the k loop). For each stream, the algorithm then loops over all the leaves that use that stream, taken in increasing order of the number of items required. For each such leaf the algorithm computes the ratio (variable *Ratio*) of cost to probability of failure of the sequence of leaves up to that leaf. The leaf with minimum such ratio is selected (leaf l_{j_0} in the algorithm, which requires d_{j_0} data items

from stream $S(l_{j_0})$). In the last loop of the algorithm, all unscheduled leaves that require d_{j_0} or fewer data items from stream $S(l_{j_0})$ are appended to the schedule in increasing order of the number of required data items.

Algorithm 1: GREEDY($\{\mathcal{L}_1, ..., \mathcal{L}_s\}, \xi, NItems$)

> **if** $\cup_{i=1}^{s} \mathcal{L}_i = \varnothing$ **then return** ξ
> $MinRatio \leftarrow +\infty$
> **for** $k = 1$ *to* s **do** loop on streams
> > $Cost \leftarrow 0$
> > $Proba \leftarrow 1$
> > $Num \leftarrow NItems[k]$
> > **for** l_j *in* $\mathcal{L}_k$ *by increasing* d_j **do**
> > > $Cost \leftarrow Cost + Proba \times (d_j - Num) \times c(k)$
> > > $Proba \leftarrow Proba \times p_j$
> > > $Num \leftarrow d_j$
> > > $Ratio \leftarrow \frac{Cost}{(1 - Proba)}$
> > > **if** $Ratio < MinRatio$ **then**
> > > > $MinRatio \leftarrow Ratio$
> > > > $j_0 \leftarrow j$
>
> **for** l_j *in* $\mathcal{L}_{S(j_0)}$ *by increasing* d_j **do**
> > **if** $d_j \le d_{j_0}$ **then**
> > > $\xi.\text{append}(l_j)$
> > > $\mathcal{L}_{S(j_0)} \leftarrow \mathcal{L}_{S(j_0)} \setminus \{l_j\}$
>
> $NItems[S(j_0)] \leftarrow d_{j_0}$
> **return** GREEDY $(\{\mathcal{L}_1, ..., \mathcal{L}_s\}, \xi, NItems)$

Theorem 1. *Algorithm 1 is optimal for the* shared *PAOTR problem for* AND*-trees.*

Proof Sketch: We prove the theorem by contradiction. We assume that there exists an instance for which the schedule produced by Algorithm 1, ξ_{greedy}, is not optimal. Among the optimal schedules, we pick a schedule, ξ_{opt}, which has the longest prefix $\mathbb{P}$ in common with ξ_{greedy}. We consider the first decision taken by Algorithm 1 that schedules a leaf that does not belong to $\mathbb{P}$. Let us denote by $l_{\sigma(1)}, ..., l_{\sigma(k)}$ the sequence of leaves scheduled by this decision. The first leaves in this sequence may belong to $\mathbb{P}$. Let $\mathbb{P}'$ be $\mathbb{P}$ minus the leaves $l_{\sigma(1)}, ..., l_{\sigma(k)}$. Then, ξ_{greedy} can be written as:

$$\xi_{greedy} = \mathbb{P}', l_{\sigma(1)}, ..., l_{\sigma(k)}, \mathbb{S}.$$

In turn, ξ_{opt} can be written $\xi_{opt} = \mathbb{P}', \mathbb{Q}, \mathbb{R}$ where $l_{\sigma(k)}$ is the last leaf of $\mathbb{Q}$. In other words, $\mathbb{Q}$ can be written $L_1, l_{\sigma(1)}, ..., L_k, l_{\sigma(k)}$, where each sequence of leaves L_i, $1 \le i \le k$, may be empty. We can write:

$$\xi_{opt} = \mathbb{P}', L_1, l_{\sigma(1)}, ..., L_k, l_{\sigma(k)}, \mathbb{R}.$$

From ξ_{greedy} and ξ_{opt}, we build a new schedule, ξ_{new},

defined as

$$\xi_{new} = \mathbb{P}', l_{\sigma(1)}, ..., l_{\sigma(k)}, L_1, ..., L_k, \mathbb{R}.$$

$\mathbb{P}', l_{\sigma(1)}, ..., l_{\sigma(k)}$ is a prefix to both ξ_{greedy} and ξ_{new}. This prefix is strictly larger than $\mathbb{P}$ since $\mathbb{P}$ does not contain $l_{\sigma(k)}$. We compute the cost of ξ_{new} and show that it is no larger than that of ξ_{opt}, thus showing that ξ_{new} is optimal and has a longer prefix in common with ξ_{greedy} than ξ_{new}, which is a contradiction. This computation is very lengthy and technical and the full proof is provided in [8]. ∎

The complexity of Algorithm 1 is $O(m^2)$. Indeed, the sets $\mathcal{L}_1, ..., \mathcal{L}_s$ are built and sorted in $O(m \log(m))$ and there are at most m recursive calls to Algorithm 1, each having a cost proportional to the number of leaves remaining in the AND tree.

One may wonder how the optimal algorithm in the *read-once* case [7], which simply sorts the leaves by increasing $d_j c(S(j))/q_j$, fares in the *shared* case. In other terms, is Algorithm 1 really needed in practice? Figure 4 shows results for a set of randomly generated AND-trees. We define the *sharing ratio*, ρ, of a tree as the expected number of leaves that use the same stream, i.e., the total number of leaves divided by the number of streams. For a given number of leaves $m = 2, \ldots, 20$ and a given sharing ratio $\rho = 1, 5/4, 4/3, 3/2, 2, 3, 4, 5, 10$, we generate 1,000 random trees for a total of 157,000 random trees (note that ρ cannot be larger than the number of leaves). Leaf success probabilities, numbers of data items needed at each leaf, and per data item costs are sampled from uniform distributions over the intervals $[0, 1]$, $[1, 5]$, and $[1, 10]$, respectively. For each tree we compute the cost achieved by the algorithm in [7] and that achieved by our optimal algorithm. Figure 4 plots these costs for all instances, sorted by increasing optimal cost. Due to this sorting, the large number of samples, and the limited resolution, the set of points for the optimal algorithm appears as a curve while the set of points for the algorithm in [7] appears as a cloud of points. The algorithm in [7] can lead to costs up to 1.86 times larger than the optimal. It leads to costs more than 10% larger for 19.54% of the instances, and more than 1% larger for 60.20% of the instances. The two algorithms lead to the same cost for 11.29% of the instances. We conclude that, in the *shared* case, Algorithm 1 provides substantial improvements over the optimal algorithm for the *read-once* case.

IV. DNF Trees

In this section we consider DNF trees. First, in Section IV-A we provide a method for computing the expected cost of a given schedule for a DNF tree. In Section IV-B we show that depth-first schedules are dominant, which means that there always exists a
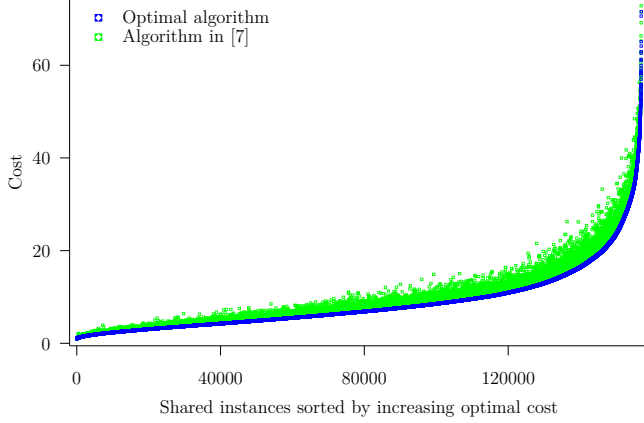
Figure 4. Cost achieved by the algorithm in [7] and that achieved by the optimal algorithm, shown for each of the 157,000 AND-tree instances sorted by increasing optimal cost.

depth-first schedule that is optimal. In Section IV-C, we then prove that the problem is NP-complete. This is in sharp contrast with the *read-once* case, in which a simple greedy algorithm is optimal [6]. In Section IV-D we propose several heuristics to schedule a DNF tree and evaluate their performance on randomly generated problem instances.

A. Evaluation of a schedule

We have seen in Section II-B in an example that computing the cost of a schedule is non-trivial for DNF trees. In this section we formalize this computation. Consider a DNF tree with N AND nodes, indexed $i = 1, \ldots, N$. AND node i has m_i leaves, denoted by $l_{i,j}$, $j = 1, \ldots, m_i$. The probability of success of leaf $l_{i,j}$ is denoted by $p_{i,j}$, and the stream that leaf $l_{i,j}$ requires is denoted by $S(i,j)$. We use $\mathcal{L}$ to denote the set of all the leaves. We consider a schedule ξ, which is an ordering of the leaves, and use $l_{s,t} \prec l_{u,v}$ to indicate that leaf $l_{s,t}$ occurs before leaf $l_{u,v}$ in ξ. We consider that the query is over s streams, S_k, $k = 1, \ldots, s$. The cost per data item of S_k is denoted by $c(S_k)$. We define the "t-th data item" of a stream as the data item produced t time-steps ago, so that the first data item is the one produced most recently, the second is the one produced before the first, etc. In this manner, when we say that a leaf $l_{i,j}$ requires $d_{i,j}$ data items it means that it requires all t-th data items of the stream for $t = 1, 2, \ldots, d_{i,j}$.

Given the above, we define $\mathcal{L}_{k,t}$ as the set of the leaves that require the t-th data item from stream S_k, and that are the first of their respective AND nodes to require that data item. Formally, we have:

$$
\mathcal{L}_{k,t} = \left\{ l_{i,j} \in \mathcal{L} \;\middle|\; \begin{array}{c} S(i,j) = S_k, \; d_{i,j} \geq t, \; \text{and} \\ \forall r \neq j, S(i,r) \neq S_k \text{ or } d_{i,r} < t \\ \text{or } l_{i,j} \prec l_{i,r} \end{array} \right\}
$$

We also define $\mathcal{A}_{i,j}$, the index set of all AND nodes that have been fully evaluated before a leaf $l_{i,j}$ is evaluated, as:

$$
\mathcal{A}_{i,j} = \{k \mid m_k = |\{l_{k,r}|l_{k,r} \prec l_{i,j}\}|\}.
$$

If we use $\mathcal{C}_{i,j,t}$ to denote the expected cost of retrieving the t-th data item of the relevant stream when evaluating leaf $l_{i,j}$, then the total cost $\mathcal{C}$ of the schedule ξ is:

$$
\mathcal{C} = \sum_{i=1}^{N} \sum_{j=1}^{m_i} \sum_{t=1}^{d_{i,j}} \mathcal{C}_{i,j,t}.
$$

The following proposition gives $\mathcal{C}_{i,j,t}$.

Proposition 2. *Given a leaf $l_{i,j}$ that requires the t-th data item from stream S_k, if there exists r such that $l_{i,r} \prec l_{i,j}$ and $l_{i,r} \in \mathcal{L}_{k,t}$, then $\mathcal{C}_{i,j,t} = 0$. Otherwise:*

$$
\mathcal{C}_{i,j,t} = \prod_{\substack{l_{r,s} \in \mathcal{L}_{k,t} \\ l_{r,s} \prec l_{i,j}}} \left(1 - \prod_{l_{r,u} \prec l_{r,s}} p_{r,u} \right)
$$

$$
\times \prod_{\substack{a \in \mathcal{A}_{i,j} \\ \nexists r, \; l_{a,r} \in \mathcal{L}_{k,t}}} \left(1 - \prod_{r=1}^{m_a} p_{a,r} \right)
$$

$$
\times \left(\prod_{l_{i,u} \prec l_{i,j}} p_{i,u} \right) \times c(S(i,j)).
$$

Proof: Consider a schedule ξ, and a leaf in that schedule, $l_{i,j}$, which requires the t-th data item from stream S_k (i.e., $S(i,j) = S_k$). Let us prove the first part of the proposition. If a leaf $l_{i,r}$ (i.e., a leaf under the same AND node as $l_{i,j}$) occurs before $l_{i,j}$ in ξ and requires the t-th item from stream S_k (i.e., $l_{i,r} \in \mathcal{L}_{k,t}$), then there are two possibilities. Either $l_{i,r}$ has been evaluated, in which case the evaluation of $l_{i,j}$ uses a data item that has already been acquired previously, hence a cost of 0. Or $l_{i,k}$ has not been evaluated, meaning that its evaluation was shortcircuited. In this case the AND node has evaluated to FALSE and the evaluation of $l_{i,j}$ is also shortcircuited, hence a cost of 0.

The second part of the proposition shows the expected cost as a product of three factors, each of which is a probability, and a fourth factor, $c(S(i,j))$, which is the cost of acquiring the data item from the stream. The interpretation of the expression for $\mathcal{C}_{i,j,t}$ is as follows: a leaf must acquire the item if and only if (i) the item has not been previously acquired; and (ii) no AND node has already evaluated to TRUE; and (iii) no leaf in the same AND node has already evaluated to FALSE. We explain the computation of these three probabilities hereafter.

The first factor is the probability that none of the leaves that precede $l_{i,j}$ in ξ and that require the t-th item from stream S_k have been evaluated. Such a leaf $l_{r,s}$ is evaluated if all the leaves in the same AND node

that precede it in the schedule have evaluated to TRUE, which happens with probability $\prod_{l_{r,u} \prec l_{r,s}} p_{r,u}$. Hence, the expression for the first factor.

The second factor is the probability that none of the AND nodes that have been fully evaluated so far has evaluated to TRUE, since if this were the case the evaluation of $l_{i,j}$ would not be needed, leading to a cost of 0. Given an AND node in $\mathcal{A}_{i,j}$, say the k-th AND node, the probability that it has been evaluated to TRUE is $\prod_{r=1}^{m_k} p_{k,r}$. This is true except if one of the leaves of that AND node belongs to $\mathcal{L}_{k,t}$. The first factor assumes that that leaf was not evaluated and, therefore, that that entire AND node was not evaluated. Hence, the expression for the second factor.

The third factor is the probability that all the leaves in the same AND as $l_{i,j}$ have evaluated to TRUE. Because we are in the second case of the proposition, none of these leaves requires the t-th item of stream S_k. All these leaves must evaluate to TRUE, otherwise the evaluation of $l_{i,j}$ would be shortcircuited, for a cost of 0. Hence, the expression for the third factor. ■

Cost of the evaluation of a schedule: To compute this cost, we need to introduce two new notations. Let $|\mathcal{L}|$ be the total number of leaves in the considered DNF, and let D be the maximum number of data items required by a stream. Then, we have $|\mathcal{L}| = \sum_{i=1}^{N} m_i$ and $D = \max_{1 \leq iN, 1 \leq j \leq m_i} d_{i,j}$.

To compute all the sets $\mathcal{L}_{k,t}$ we need to scan the leaves of each AND node according to the schedule ξ while recording the maximum number of elements required from each stream. The overall cost of this scheme is $O(|\mathcal{L}|)$. Each set $\mathcal{L}_{k,t}$ contains at most N leaves.

Computing all the sets $\mathcal{A}_{i,j}$ is also done through a traversal of the set of leaves, for an overall cost of $O(|\mathcal{L}| + N^2)$ (because the sets $\mathcal{A}_{i,j}$ take at most $N - 1$ distinct values and each contains at most $N - 1$ elements). Computing all the product of probabilities used in the computation of all the $\mathcal{C}_{i,j,t}$ can also be done in a single traversal of the set of leaves.

Once all these precomputations are done, the first term in the expression of $\mathcal{C}_{i,j,t}$ can be computed in $O(N)$ and the second in $O(N^2)$, and the third one in $O(1)$. Overall the cost of a schedule can be evaluated with complexity

$$O(|\mathcal{L}|DN^2).$$

B. Dominance of depth-first schedules

Theorem 2. *Given a* DNF *tree, there exists an optimal schedule that is depth-first, i.e., that processes* AND *nodes one by one.*

Proof: Consider a DNF tree $\mathcal{T}$ and a schedule ξ. Without loss of generality we assume that the AND nodes, $A_1, \ldots, A_n$, are numbered in the order of their completion. Thus, according to ξ, A_1 is the first AND node with all its leaves evaluated. We denote by M the number (possibly zero) of AND nodes that ξ processes one by one and entirely at the start of its execution. Therefore, if ξ evaluates a leaf $l_{i,j}$, with $i \neq 1$, in the m_1 first steps, then $M = 0$. Finally, we assume that the leaves of an AND node are numbered according to their evaluation order in ξ.

We prove the theorem by contradiction. Let us assume that there does not exist a schedule that satisfies the desired property. Let ξ be an optimal schedule that maximizes M. By definition of M and by the hypothesis on the numbering of the AND nodes, schedule ξ evaluates some leaves of the AND nodes $A_{M+2}, ..., A_n$ before it evaluates the last leaf of A_{M+1}. Let $\mathcal{L}$ denote the set of these leaves. We now define a new ξ' which starts by executing at least $M + 1$ AND nodes one by one:

- ξ' starts by evaluating the first M AND nodes one by one, evaluating their leaves in the same order and at the same steps as in ξ;
- ξ' then evaluates all the leaves of A_{M+1} in the same order as in ξ (but not at the same steps);
- ξ' then evaluates the leaves in $\mathcal{L}$ in the same order as in ξ (but not at the same steps);
- ξ' finally evaluates the remaining leaves in the same order and at the same steps as in ξ.

The cost of a schedule is the sum, over all potentially acquired data items, of the cost of acquiring each data item times the probability of acquiring it. Let d be a data item potentially needed by a leaf in $\mathcal{T}$. We show that the probability of acquiring d is not greater with ξ' than with ξ. We have three cases to consider.

Case 1) d is not needed by a leaf of A_{M+1} and not needed by a leaf in $\mathcal{L}$. Then d's probability to be acquired is the same with ξ and ξ'.

Case 2) d is needed by at least one leaf of A_{M+1}. The only way in which a leaf that is evaluated in ξ would not be evaluated in ξ' is if A_{M+1} evaluates to TRUE. By assumption, however, at least one leaf of A_{M+1} uses d. Therefore, for A_{M+1} to evaluate to TRUE, d must be acquired. Consequently, the probability that d is acquired is the same with ξ and with ξ'.

Case 3) d is needed by at least one leaf in $\mathcal{L}$ but not needed by any leaf of A_{M+1}. ξ and ξ' define the same ordering on the leaves in $\mathcal{L}$. For each AND node A_i, with $M + 2 \leq i \leq N$, there is at most one leaf in $A_i \cap \mathcal{L}$ that can be the leaf responsible for the acquisition of d with ξ, and it is the same leaf with ξ'. Let $\mathcal{F}$ be the set of all these leaves. Then, with ξ, the leaves in $\mathcal{F}$ are responsible for the acquisition of d if and only if:

- $A_1, ..., A_M$ all evaluate to FALSE;
- None of the evaluated leaves of $A_1, ..., A_M$ needs d; and
- At least one of the leaves in $\mathcal{F}$ is evaluated.

Let us denote by $\mathcal{P}$ the probability that all the AND nodes A_1, ..., A_M evaluate to FALSE and that none of the evaluated leaves of these AND nodes needs the data item d. Let us denote by $\mathcal{D}$ the probability that d is acquired because of the evaluation of one of the leaves of the AND nodes A_1, ..., A_M. Finally, let $\mathcal{R}$ be the probability that one of the leaves evaluated with ξ after $l_{M+1,m_{M+1}}$ acquires d, knowing that no leaves of A_1, ..., A_M or in $\mathcal{L}$ acquires it. Then, with ξ, the probability p that d is acquired is:

$$p = \mathcal{D} + \mathcal{P}\left(1 - \prod_{l_{i,j} \in \mathcal{F}}\left(1 - \prod_{k=1}^{j-1} p_{i,k}\right)\right) + \mathcal{R} \quad (1)$$

because leaf $l_{i,j}$ is evaluated with probability $\prod_{k=1}^{j-1} p_{i,k}$, that is, if all the leaves from the same AND node that are evaluated prior to it all evaluate to TRUE. The second term of Equation (1) is the probability that the leaves in $\mathcal{F}$ are responsible for acquiring d.

With schedule ξ', the leaves of $\mathcal{F}$ are responsible for the acquisition of d if and only if:
- The AND nodes A_1, ..., A_M, and A_{M+1} all evaluate to FALSE;
- None of the evaluated leaves of the AND nodes A_1, ..., A_M need d; and
- At least one of the leaves in $\mathcal{F}$ is evaluated.

Thus, with ξ', the probability p' that d is acquired is:

$$p' = \mathcal{D} + \mathcal{P}\left(1 - \prod_{k=1}^{m_{M+1}} p_{M+1,k}\right)$$
$$\times \left(1 - \prod_{l_{i,j} \in \mathcal{F}}\left(1 - \prod_{k=1}^{j-1} p_{i,k}\right)\right) + \mathcal{R}$$

Comparing this equation with Equation 1, we see that p' is not greater than p.

The probability that a data item is acquired with ξ' is thus not greater than with ξ. Therefore, in each of the three cases the cost of ξ' is not greater than the cost of ξ, meaning that ξ' is also an optimal schedule. Since ξ' starts by executing at least $M + 1$ AND nodes one by one, we obtain a contradiction with the maximality assumption on M, which concludes the proof. ∎

C. NP-completeness

In the *read-once* case, an optimal algorithm for DNF trees is built on top of the optimal algorithm for AND-trees [6]. The same approach cannot be used in the *shared* case, as seen in a simple counter-example [8]. In other words, for some DNF trees, the ordering of the leaves of a given AND node in an optimal schedule does not correspond to the ordering produced by Algorithm 1 for that AND node. And, in fact, in this section we show

the NP-completeness of finding an optimal schedule to evaluate a DNF tree.

Definition 1 (DNF-Decision)**.** *Given a* DNF *tree and a cost bound K, is there a schedule whose expected cost does not exceed K?*

Theorem 3. DNF-Decision *is NP-complete.*

Proof: The NP-completeness is obtained via a non-trivial reduction from 2-PARTITION [9]. See [8] for the proof. ∎

D. Heuristics

Given the NP-completeness result in the previous section, we now propose several polynomial-time heuristics for computing a schedule. These heuristics fall into three categories, which we term leaf-ordered, AND-ordered, and stream-ordered.

Leaf-ordered heuristics simply sort the leaves according to leaf costs ($\mathcal{C}$), failure probabilities ($q = 1 - p$), or the ratio of the two, which leads to three heuristics plus a baseline random one:
- Leaf-ordered, decreasing q (prioritizes leaves with high chances of shortcutting the evaluation of an AND node);
- Leaf-ordered, increasing $\mathcal{C}$ (prioritizes leaves with low costs);
- Leaf-ordered, increasing $\mathcal{C}/q$ (prioritizes leaves with low costs and also with high chances of shortcutting the evaluation of an AND node);
- Leaf-ordered, random (baseline).

The above first three heuristics have intuitive rationales. Other options are possible (e.g., sort leaves by decreasing $\mathcal{C}$) but are easily shown to produce poor results.

AND-ordered heuristics, unlike leaf-ordered heuristics, account for the structure of the DNF tree by building depth-first schedules, with the rationale that there is a depth-first schedule that is optimal (Theorem 2). Furthermore, Algorithm 1 provides a way to compute an optimal schedule for the leaves within the same AND node. For this optimal schedule one can compute the (expected) cost and the probability of success of the AND node using the method in Section IV-A. Therefore, AND-ordered heuristics simply order the AND nodes based on their computed costs ($\mathcal{C}$), computed probability of success (p), or ratio of the two, and using Algorithm 1 for scheduling the leaves of each AND node, leading to three heuristics:
- AND-ordered, decreasing p (prioritizes AND's with high chances of shortcircuiting the evaluation of the OR node);
- AND-ordered, increasing $\mathcal{C}$ (prioritizes AND's with low costs);

- AND-ordered, increasing $\mathcal{C}/p$ (prioritizes AND's with low costs and also with high chances of short-circuiting the evaluation of the OR node);

There are two approaches to compute the cost of an AND node: (i) consider the AND node in isolation assuming that the OR node has a single AND node child; or (ii) account for previously scheduled AND nodes whose evaluation has caused some data items to be acquired with some probabilities. We terms the first approach "static" and the second approach "dynamic," giving us two versions of the last two heuristics above.

Stream-ordered heuristics proceed by ordering the streams from which data items are acquired, acquiring all items from a stream before proceeding to the next stream, until the truth value of the OR node has been determined. This idea was proposed in [4], and to the best of our knowledge it is the only previously proposed heuristic for solving the PAOTR problem in the *shared* scenario for DNF trees. For each stream S the heuristic computes a metric, $R(S)$, defined as follows:

$$R(S) = \frac{\sum_{i,j|S(i,j)=S} q_{i,j} n_{i,j}}{\max_{i,j|S(i,j)=S} d_{i,j} c(S)},$$

where $n_{i,j}$ is the number of leaf nodes whose evaluation would be shortcircuited if leaf $l_{i,j}$ was to evaluate to FALSE. The numerator can thus be interpreted as the shortcutting power of stream S. The denominator is the maximum data element acquisition cost over all the leaves that use stream S. The heuristic orders the streams by increasing R values. The rationale is that one should prioritize streams that can shortcut many leaf evaluations and that have low maximum data item acquisition costs. The heuristic as it is described in [4] acquires the maximum number of needed data items from each stream so as to compute truth values of all the leaves that require data items from that stream. In other words, the leaves that require data items from the same stream are scheduled in decreasing $d_{i,j}$ order. However, Proposition 1 holds for DNF trees, showing that it is always better to schedule these leaves in *increasing $d_{i,j}$* order. We use this leaf order to implement this heuristic in this work. We have verified in our experiments that this version outperforms the version in [4] in the vast majority of the cases, with all remaining cases being ties.

In total, we consider 4 leaf-ordered, 5 AND-ordered, and 1 stream-ordered heuristics. We first evaluate these heuristics on a set of "small" instances for which we can compute optimal schedules using an exponential-time algorithm that performs an exhaustive search. Such an algorithm is feasible because, due to Theorem 2, it only needs to search over all possible depth-first schedules. Small instances are generated using the same method as that described in Section III-B for generating AND-tree instances. We generate DNF
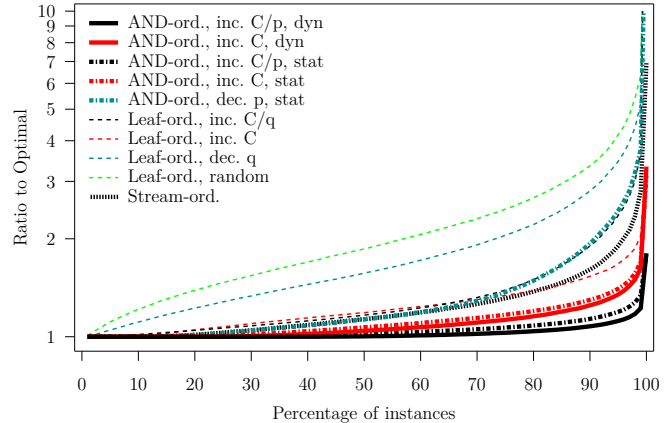

Figure 5. Ratio to optimal vs. fraction of the instances for which a smaller ratio is achieved, computed over the 21,600 "small" DNF tree instances.

trees with $N = 2, \ldots, 9$ AND nodes and up to at most 20 leaves and 8 leaves per AND, generating 100 random instances for each configuration, for a total of 21,600 instances (The source code is available at www.ens-lyon.fr/LIP/ROMA/Data/DataForRR-8373.tgz). For each instance we compute the ratio between the cost achieved by each heuristic and the optimal cost. Figure 5 shows for each heuristic the ratio vs. the fraction of the instances for which the heuristic achieves a lower ratio. For instance, a point at $(80, 2)$ means that the heuristic leads to schedules that are within a factor 2 of optimal for 80% of the instances, and more than a factor 2 away from optimal for 20% of the instances. The better the heuristic the closer its curve is to the horizontal axis.

The trends in Figure 5 are clear. Overall the poorest results are achieved by the leaf-ordered heuristics, with the random such heuristic expectedly being the worst and the increasing $\mathcal{C}$ the best. The AND-ordered heuristics, save for the decreasing p version, lead to the best results overall. More precisely, the best results are achieved by sorting AND's by increasing $\mathcal{C}/p$, with sorting by increasing $\mathcal{C}$ leading to the second-best results. For the two AND-ordered heuristics that have both a static and a dynamic version, the dynamic version leads to marginally better results than the static version. Finally, the stream-ordered heuristic leads to poorer results than the best leaf-ordered heuristics, and thus significantly worse than the best AND-ordered heuristics.

We also evaluate the heuristics on a set of "large" instances with $N = 2, \ldots, 10$ AND nodes and $m = 5, 10, 15, 20$ leaves per AND node, with 100 random instances per configuration, for a total of 32,400 instances. For most of these instances we cannot tractably compute the optimal cost. Consequently, we compute ratios to the cost achieved by the AND-ordered by increasing $\mathcal{C}/p$ dynamic heuristic, which leads to the best
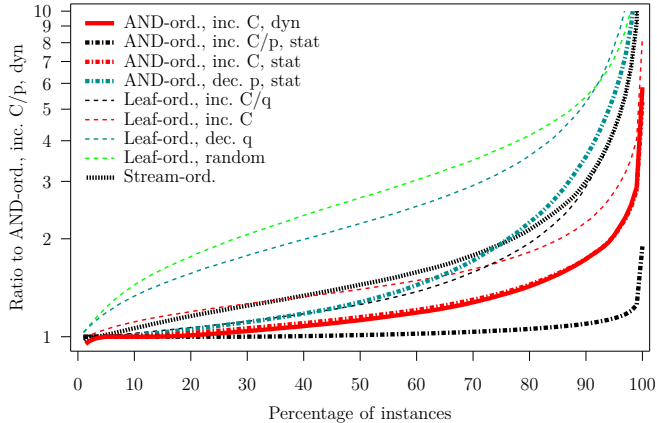
Figure 6. Ratio to AND-ordered increasing $\mathcal{C}/p$ dynamic vs. fraction of the instances for which a smaller ratio is achieved, computed over the 32,400 "large" DNF tree instances.

results for small instances. Results are shown in Figure 6. Essentially, all the observations made on the results for small instances still hold. We conclude that the best approach is to build a depth-first schedule, to sort the AND nodes by the ratio of their costs to probability of success, and to compute these costs dynamically, accounting for previously scheduled AND nodes. This heuristic is the best one in 94.5%, resp. 83.8%, of the cases reported in Figure 6, resp. Figure 5. It runs in less than 5 seconds on a 1.86 GHz core when processing a tree with 10 AND nodes with each 20 leaves.

V. Conclusion

Motivated by a query processing scenario for sensor data streams, we have studied a version of the Probabilistic And-Or Tree Resolution (PAOTR) problem [6] in which a data stream can be referenced by multiple leaves. We have given an optimal algorithm in the case of AND-trees and have shown NP-completeness in the case of DNF trees. For DNF we have shown that there is an optimal solution that corresponds to a depth-first traversal of the tree. This observation provides inspiration for a heuristic that largely outperforms the heuristic previously proposed in [4].

A possible future direction is to consider so-called *non-linear strategies* [6]. Although in this work we have considered a schedule as a leaf ordering (called a *linear strategy* in [6]), a more general notion is that of a decision tree in which the next leaf to be evaluated is chosen based on the truth value of the previous evaluated leaf. A practical drawback of a non-linear strategy is that the size of the strategy's description is exponential in the number of tree leaves. In [6], it is shown that in the *read-once* case linear strategies are dominant for DNF trees, meaning that there is always one optimal strategy that is linear. Via a simple counter example it can be shown that this is

no longer true in the *shared* case [8], thus motivating the investigation of non-linear strategies. Another possible future direction is to consider a less restricted version of the problem in which a single predicate at a leaf can access multiple streams rather than just a single one (e.g., "$AVG(X, 10) < 10 \geq MIN(Y, 20)$"). There is no reason for real-world queries to be limited to a single stream per predicate. An interesting question, then, is whether the PAOTR problem remains polynomial for AND-trees or whether it becomes NP-complete.

References

[1] E. Miluzzo, "Sensing Meets Mobile Social Networks: The Design, Implementation and Evaluation of the CenceMe Application," in *Proc. of ACM Conf. on Embedded Networked Sensor Systems*, 2008.

[2] I. Mohomed, A. Misra, M. Ebling, and W. Jerome, "Context-Aware and Personalized Event Filtering for Low-Overhead Continuous Remote Health Monitoring," in *Proc. of the IEEE Intl. Symp. on a World of Wireless Mobile and Multimedia Networks*, 2008.

[3] S. Gaonkar, J. Li, R. Roy Choudhury, L. Cox, and A. Schmidt, "Micro-Blog: Sharing and Querying Content through Mobile Phones and Social Participation," in *Proc. of the ACM Intl. Conf. on Mobile Systems, Applications, and Services*, 2008.

[4] L. Lim, A. Misra, and T. Mo, "Adaptive Data Acquisition Strategies for Energy-Efficient Smartphone-based Continuous Processing of Sensor Streams," *Distributed Parallel Databases*, vol. 31, no. 2, pp. 321–351, 2013.

[5] "The SHIMMER sensor platform," http://shimmer-research.com, 2013.

[6] R. Greiner, R. Hayward, M. Jankowska, and M. Molloy, "Finding Optimal Satisficing Strategies for And-Or Trees," *Artificial Intelligence*, vol. 170, no. 1, pp. 19–58, 2006.

[7] D. E. Smith, "Controlling backward inference," *Artificial Intelligence*, vol. 39, no. 2, pp. 145—208, 1989.

[8] H. Casanova, L. Lim, Y. Robert, F. Vivien, and D. Zaidouni, "Cost-Optimal Execution of Trees of Boolean Operators with Shared Streams," Inria, Research Report RR-8373, 2013, http://hal.inria.fr/hal-00869340.

[9] M. R. Garey and D. S. Johnson, *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.