Cost-Optimal Execution of Boolean DNF Trees with Shared Streams

Henri Casanova, Lipyeow Lim, Yves Robert, Frédéric Vivien, Dounia Zaidouni

Cost-Optimal Execution of Boolean DNF Trees with Shared Streams

Henri Casanova, Lipyeow Lim, Yves Robert, Frédéric Vivien, and Dounia Zaidouni

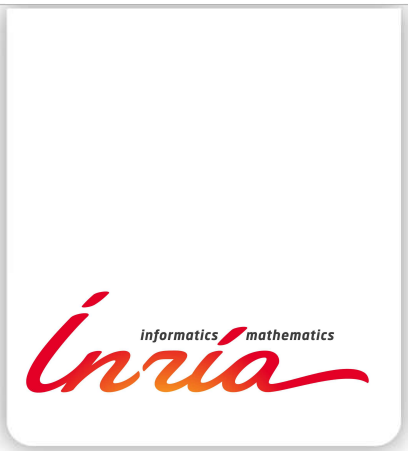
RESEARCH REPORT

N° 8616

December 2014

Project-Team Roma

ISSN 0249-6399 ISRN INRIA/RR--8616--FR+ENG



Cost-Optimal Execution of Boolean DNF Trees with Shared Streams

Henri Casanova*, Lipyeow Lim*, Yves Robert†‡, Frédéric Vivien§‡, and Dounia Zaidouni§‡

Project-Team Roma

Research Report n° 8616 — December 2014 — 53 pages

Abstract: Several applications process queries expressed as trees of Boolean operators applied to predicates on sensor data streams, e.g., mobile apps and automotive apps. Sensor data must be retrieved from the sensors, which incurs a cost, e.g., an energy expense that depletes the battery of a mobile device, a bandwidth usage. 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 study the case in which a data stream occurs in multiple predicates, either when each predicate references a single stream or when a predicate can reference multiple streams. In the single-stream case we give an optimal algorithm for a single-level tree and show that the problem is NP-complete for DNF trees. For DNF trees we show that there exists an optimal predicate evaluation order that is depth-first, which provides a basis for designing a range of heuristics. In the multi-stream case we show that the problem is NP-complete even for single-level trees. As in the single stream case, for DNF trees we show that there exists a depth-first leaf evaluation order that is optimal and we design efficient heuristics.

Key-words: query processing, boolean operators, energy, scheduling, greedy algorithm, data sharing

* University of Hawaii at Mānoa, HI, USA
† École normale supérieure de Lyon, France
‡ LIP laboratory – CNRS, ENS Lyon, INRIA, UCB Lyon 1
§ INRIA

RESEARCH CENTRE GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Exécution de coût optimal d'arbres d'opérateurs booléens partageant des données

Résumé : Le traitement de requêtes, exprimées sous forme d'arbres d'opérateurs booléens appliqués à des prédicats sur des flux de données de senseurs, a de nombreuses applications dans le domaine du calcul mobile. Les données doivent être transférées des senseurs vers l'appareil de traitement des données, ce qui induit un coût, par exemple une consommation énergétique qui diminue la charge des batteries, ou une utilisation de bande passante. L'objectif est de déterminer l'ordre dans lequel les prédicats doivent être évalués de manière à court-circuiter une partie de l'évaluation de l'arbre et de minimiser l'espérance du coût. Ce problème a déjà été étudié sous l'hypothèse que chaque flux apparaît dans un seul prédicat. Dans le présent travail nous étudions le cas où un flux peut être utilisé par plusieurs prédicats, soit quand chaque prédicat référence un unique flux (cas *mono-flux*), soit quand un prédicat peut référencer plusieurs flux (cas *multi-flux*). Dans le cas mono-flux, nous donnons un algorithme optimal pour les arbres à un seul niveau et nous montrons que le problème est NP-complet pour les arbres sous forme normale disjonctive (FND). Pour les arbres FND nous montrons qu'il existe un ordre optimal d'évaluation des prédicats qui est en profondeur d'abord; cette propriété nous sert à proposer toute une gamme d'heuristiques. Dans le cas multi-flux, nous montrons que le problème est NP-complet même pour les arbres à un seul niveau. Comme dans le cas mono-flux, nous montrons qu'il existe un ordre d'évaluation des prédicats en profondeur d'abord qui est optimal et nous proposons des heuristiques efficaces.

Mots-clés : traitement de requêtes, opérateurs booléens, énergie, ordonnancement, algorithmique probabiliste, algorithme glouton, partage de données

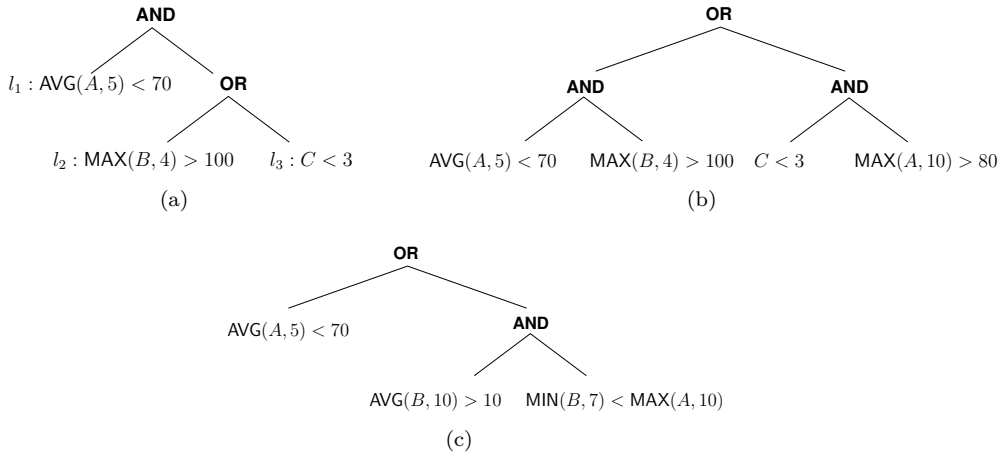


Figure 1: Three query tree examples: (a) a *read-once* query; (b) a *shared* query in the *single-stream* case; (c) a *shared* query in the *multi-stream* case.

1 Introduction

An increasing number of applications are being developed or envisioned that continuously process data generated via sensors embedded in or associated with mobile devices. 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 for, e.g., social networking [1], remote health monitoring [2]. Automotive applications running on a smartphone can acquire data from sensors in a vehicle (e.g., engine status, speed, angular speed) as well as from remote databases (e.g., weather, traffic, road works) so as to perform continuous queries that trigger appropriate responses (e.g., alerting the driver that driving conditions are dangerous) [3].

In the above applications there is a cost associated to the acquisition of sensor data. Even moderate data rates can cause commercial smartphone batteries to be depleted in a few hours [4]. In the automotive application scenario, the acquisition of sensor data incurs a cost in terms of bandwidth usage on the sensor network in the vehicle [3]. Consequently, solutions must be developed to reduce the cost of sensor data acquisition when processing continuous queries.

In this work we study the problem of minimizing the (expected) cost of sensor data acquisition when evaluating a query expressed as a tree of conjunctive and disjunctive Boolean operators applied to Boolean *predicates* on the data. Each predicate is computed over data items from different data streams generated periodically by sensors, and has 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 [5], by which the query engine carefully chooses the *order* and the *numbers of data items* to acquire 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 sensor platforms [6, 3].

Three example query trees are shown in Figure 1, assuming streams named A, B, and C,

which 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, MIN, 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), or multiple operators (e.g., "$\mathsf{MIN}(B, 7) < \mathsf{MAX}(A, 10)$").

Most results in the literature are for *read-once* queries, i.e., when each data stream occurs in at most one leaf of the query tree. The example query tree in Figure 1(a) is a *read-once* query since no stream occurs in two leaves. In this work we study the more general case, which we term *shared*, in which a stream can occur in multiple leaves. Figure 1(b) shows a *shared* query in which stream A occurs at two leaves. Such queries are easily envisioned in most domains. For instance, in a telehealth example, an alert may be generated either if the heart rate is high and the acceleration is zero or if the heart rate is low and the SPO2 (blood oxygen saturation) is low. *Shared* queries relevant to an automotive application are considered in [3]. We also study the case in which multiple streams can occur in a single predicate. An example is shown in Figure 1(c), in which streams A and B occur in multiple leaves, and together in one leaf. We term the scenarios in Figure 1(a) and 1(b) *single-stream* and the scenario in Figure 1(c) *multi-stream*.

Considering *shared* queries has important algorithmic implications that we explore in this work. The device that processes the query acquires data items from streams and holds each data item in memory until the query has been processed. 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 operators 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 acquiring 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 acquired.

In this work we study the *shared* queries and make the following contributions:

- For AND query trees:
  - We give a polynomial-time greedy algorithm (which is not as straightforward as the optimal algorithm for *read-once* queries) that is optimal in the *single-stream* case, and show that the problem in NP-complete in the *multi-stream* case.
  - For the *multi-stream* case we propose an extension of the *single-stream* greedy algorithm. This extension is not optimal but computes near-optimal leaf evaluation orders in practice.
- For DNF query trees:
  - We show that the problem is NP-complete in the *single-stream* case (and thus also in the *multi-stream* case).
  - In both the *single-stream* and *multi-stream* case we show that there exists an optimal leaf evaluation order that is depth-first;
  - We develop heuristics that we evaluate in simulation and compare to the optimal solution (computed via an exhaustive search on small instances) and to the *single-stream* heuristic proposed in [5].

Section 2 discusses related work. Section 3 defines the problem and our assumptions formally. Section 4 gives a method for computing the expected cost of a leaf evaluation order. We then study the problem for AND trees and DNF trees in Section 5 and Section 6, respectively, for both the *single-stream* and *multi-stream* cases. Section 7 concludes the paper with a brief summary of our findings and perspectives on future work. The source code and data for all experiments are publicly available[1].

[1] `https://github.com/henricasanova/AISubmission_12_9_2014`

2 Related work

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.

For *read-once* queries the complexity of the PAOTR problem is unknown for general AND-OR trees. Smith et al. [7] propose a simple $O(n \log n)$ greedy algorithm (n is the number of leaves in the query tree) that produces an optimal leaf evaluation order for AND trees (i.e., single-level trees with an AND operator at the root node). Greiner et al. [8] survey known theoretical results and present several new results. They consider a depth-first approach that recursively replaces rooted subtrees with a single equivalent single node. They show that this approach can be arbitrarily sub-optimal for trees with 3 levels or more. For DNF trees (i.e., collections of AND trees whose roots are the children of a single OR node), they show that this approach is dominant, meaning that there is always one optimal strategy that corresponds to a depth-first traversal. They proposed a $O(n \log n)$ depth-first traversal of the tree that reuses the algorithm in [7] to order leaves within each AND, which produces an optimal evaluation order for any DNF tree.

Shared queries, the focus of this work, are important in practice and have been introduced and investigated by Lim et al. [5]. 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 for *shared* queries 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, interdependences 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 for *read-once* queries.

Several problems studied in the literature are closely related to the PAOTR problem and fall in the area of system testing [9], and in particular the evaluation of priced functions [10] and the Discrete Function Evaluation Problem (DFEP) [11].

Charikar et al. [10] and Cicalese et al. [12] study the evaluation of "priced functions." In this context an algorithm seeks to read a subset of the function's inputs so that the function's output can be determined with minimum price. When the function is an AND/OR tree there exist algorithms that achieve the best possible competitive ratio in pseudo-polynomial [10] and polynomial [12] time. In this context, no knowledge of the Boolean variables (which correspond to our predicates) is assumed. In the context of our work this would correspond to all predicates having the same probability of success of $\frac{1}{2}$. However, ignoring probabilities of success can lead to solutions that are no better than L-competitive for functions with L inputs, even for a single AND tree[2].

Cicalese et al. [11] have studied the Discrete Function Evaluation Problem (DFEP). An instance of DFEP is defined by a set S of objects, a partition C of these objects into classes (which represent the values taken by the function), a probability distribution p on S, a set T of tests, and a cost function assigning a cost to each test. The goal of DFEP is to design a testing procedure that uses tests from T to identify the class that includes the unknown object, while

[2]Consider an AND node with L leaves of unit cost, $L-1$ having a probability of success of $1-\epsilon$ and the last one having a probability of success of ϵ. If the ϵ-probability leaf is evaluated first the expectation of the cost of the evaluation of the query is $1+\epsilon(1+(1-\epsilon)(1+(1-\epsilon)(...+(1-\epsilon)1))) = 1+\epsilon((1-\epsilon)^0+(1-\epsilon)^1+...+(1-\epsilon)^{L-2}) = 1+(1-(1-\epsilon)^{L-1}) \approx 1+(L-1)\epsilon$ when ϵ tends to zero. When probabilities are ignored, all leaves are absolutely equivalent and a probability-agnostic leaf evaluation order can evaluate the ϵ-probability leaf last, leading to a leaf evaluation order with expected cost equal to: $1+(1-\epsilon)(1+(1-\epsilon)(1+...(1+(1-\epsilon)(1)))) = (1-\epsilon)^0+(1-\epsilon)^1+...+(1-\epsilon)^{L-1} = \frac{1-(1-\epsilon)^L}{1-(1-\epsilon)} \approx L$. Hence the lower bound on the competitive ratio.

minimizing the expectation of the testing cost (or the worst-case testing cost). DFEP has also been studied under the names of the Equivalence Class Determination problem [13] and of the Group Identification problem [14].

Evaluation of AND/OR trees is a special case of DFEP where the set of objects is the set of the possible instantiations of the vector of predicates, where the probability distribution is the probability of occurrences of the different instantiations, where there are only two classes corresponding to instantiations leading to the AND/OR tree evaluating to TRUE or FALSE, and where tests correspond to the evaluation of predicates. There exist approximation algorithms for the minimization of the expectation of the testing cost with ratio $O(\log(|S|))$ [11] or $O\left(\log\left(\frac{1}{p_{min}}\right)\right)$ [13, 14], where p_{min} is the minimum probability of an object ($p_{min} = \min_{s \in S} p(s)$). For AND/OR trees, $|S| = 2^L$ and $p_{min} \leq \left(\frac{1}{2}\right)^L$. Therefore, all these approximations algorithms are $O(L)$-approximation algorithms for AND/OR tree evaluation. However, for *single-stream* AND/OR trees without reuse where all costs are identical (what is often called the uniform case), any schedule is a $O(L)$-approximation if the AND/OR tree is neither a tautology nor a false statement: in the best case at least one predicate must be evaluated, and in the worst case all L predicates must be evaluated. Therefore, existing results for DFEP do not lead to efficient solutions for the AND/OR tree evaluation problem.

3 Problem statement / Framework

A query is an AND-OR *tree*, i.e., a rooted tree whose non-leaf nodes are AND or OR operators, and whose leaves are labeled with probabilistic Boolean predicates. Each predicate is evaluated over data items generated by data *streams*. The evaluation of each predicate has a known *success probability*, i.e., the probability that the predicate evaluates to TRUE. In practice, the success probability can be estimated based on historical traces obtained from previous query evaluations. As in [8], we assume *independent* predicates, meaning that two predicates at two leaves in a query are statistically independent. Evaluating a predicate incurs has a *cost* determined by the number of data items required to perform the evaluation and a per data item cost 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 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 L leaves $\mathcal{L} = \{l_1, \ldots, l_L\}$. Leaf l_j has success, resp. failure, probability p_j, resp. $q_j = 1 - p_j$, and requires the last $d_{l_j}^{s_k}$ items from each stream $s_k \in \mathcal{S}$. $d_{l_j}^{s_k}$ is zero if l_j does not require items from s_k. 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 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 leaf sequence.

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. 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 corresponds to a *read-once* query.

The PAOTR problem consists in determining a schedule with minimum cost. For *read-once* queries the complexity of PAOTR is unknown for general AND-OR query trees, while optimal polynomial-time algorithms are known for AND trees [7] and DNF trees [8]. In this work, we focus on these two types of trees as well but for *shared* queries.

4 Evaluation of a schedule

Our overall objective is to study the problem of computing an optimal schedule for AND and DNF trees for *shared* queries. First, in this section we explain how the expected cost of a given schedule can be computed. This computation is non-trivial, as seen on an example (Section 4.1), but can be performed in polynomial time (Section 4.2).

4.1 Schedule evaluation example

In this section, we illustrate on an example what is involved when computing the expected cost of a schedule. Consider the DNF tree in Figure 2 with three AND nodes, for four streams A, B, C, and D. For each leaf we indicate how many data items it requires from each stream and its probability of success. In this example each leaf requires a single data item from a stream. Since each leaf requires data items from a single stream this tree corresponds to the *single-stream* case. We provide a (more complex) example for the *multi-stream* case in A. Leaves are labeled l_1 to l_7, in the order in which they appear in a given schedule. Computing the cost of a schedule is much more complicated than for *read-once* queries 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 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: $\mathcal{C}_6 = 0$.

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, requires the data item that is needed by l_7 from stream D. Therefore, l_7 must be evaluated and its evaluation

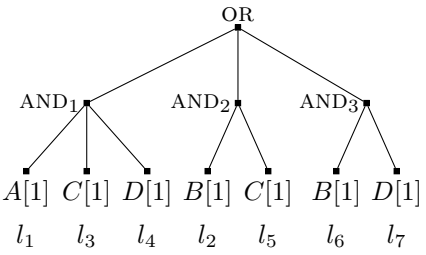


Figure 2: Example *single-stream* DNF tree.

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, $\mathcal{C}_7 = (1 - p_1p_3)(1 - p_2p_5)p_6c(D)$.

Overall, we obtain the cost of the schedule:

$$\mathcal{TC} \quad = \quad c(A) + c(B) + (p_1 + (1 - p_1)p_2)c(C) + (p_1p_3 + (1 - p_1p_3)(1 - p_2p_5)p_6)c(D).$$

Given the complexity of the above cost computation on a small example, one might expect the PAOTR problem to be NP-complete for *shared* queries (recall that it is polynomial for *read-once* queries). We confirm this expectation in Section 6.

4.2 Schedule evaluation algorithm

Consider a DNF tree with N AND nodes, indexed $i = 1, \ldots, N$. As defined in Section 3 the set of leaves is denoted by $\mathcal{L}$ and has cardinality L. To capture the structure of the DNF tree we modify the leaf notation in Section 3 as follows. 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 $p_{i,j}$. The query is over S streams, s_s, $s = 1, \ldots, S$ and each leaf can require data from multiple streams as in the more general *multi-stream* case. The cost per data item of s_s is denoted by $c(s_s)$. 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 leaf $l_{i,j}$ requires $d^{s_s}_{l_{i,j}}$ data items from stream s_s it means that it requires all t-th data items of the stream s_s for $t = 1, 2, \ldots, d^{s_s}_{l_{i,j}}$. Finally, we consider a schedule ξ, which is an ordering of the leaves, and use $l_{r,t} \prec l_{u,v}$ to indicate that leaf $l_{r,t}$ occurs before leaf $l_{u,v}$ in ξ.

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

$$\mathcal{L}_{s,t} = \left\{ l_{i,j} \in \mathcal{L} \;\middle|\; d^{s_s}_{l_{i,j}} \geq t, \text{ and } (\forall r \neq j, d^{s_s}_{l_{i,r}} < t \text{ or } l_{i,j} \prec l_{i,r}) \right\}.$$

We also define $\mathcal{A}_{i,j}$, the index set of all AND nodes that have been fully evaluated before 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 $\mathcal{C}_{l_{i,j},s,t}$ to denote the expected cost of retrieving the t-th data item of stream s_s when evaluating leaf $l_{i,j}$, then the total cost $\mathcal{C}$ of the schedule ξ is:

$$\mathcal{C} = \sum_{l_{i,j} \in \xi} \left(\sum_{s=1}^{S} \sum_{t=1}^{d^{s_s}_{l_{i,j}}} \mathcal{C}_{l_{i,j},s,t} \right). \tag{1}$$

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

Proposition 1. *Given a leaf $l_{i,j}$ that does not require the t-th data item from stream s_s, then $\mathcal{C}_{l_{i,j},s,t} = 0$. Otherwise, if there exists r such that $l_{i,r} \prec l_{i,j}$ and $l_{i,r} \in \mathcal{L}_{s,t}$, then $\mathcal{C}_{l_{i,j},s,t} = 0$, else:*

$$\begin{aligned}\mathcal{C}_{l_{i,j},s,t} = & \prod_{\substack{l_{r,v}\in\mathcal{L}_{s,t}\\ l_{r,v}\prec l_{i,j}}} \left(1 - \prod_{l_{r,u}\prec l_{r,v}} p_{r,u}\right) \\ \times & \prod_{\substack{a\in\mathcal{A}_{i,j}\\ \nexists r,\ l_{a,r}\in\mathcal{L}_{s,t}}} \left(1 - \prod_{r=1}^{m_a} p_{a,r}\right) \\ \times & \left(\prod_{l_{i,u}\prec l_{i,j}} p_{i,u}\right) \times c(s_s).\end{aligned}$$

Proof. Consider a schedule ξ, a stream s_s, and an integer t. Consider a leaf in that schedule, $l_{i,j}$, which requires the t-th data item from stream s_s. Let us prove the first part of the proposition. If leaf $l_{i,j}$ does not require the t-th data item from stream s_s, then the acquisition cost is 0. Otherwise, if a leaf $l_{i,r}$ (i.e., a leaf in the same AND node as $l_{i,j}$) occurs before $l_{i,j}$ in ξ ($l_{i,r} \prec l_{i,j}$) and requires the t-th item from stream s_s (i.e., $l_{i,r} \in \mathcal{L}_{s,t}$), 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 and the cost is 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, $c(s_s)$, which is the cost of acquiring the data item from the stream. The interpretation of the expression for $\mathcal{C}_{l_{i,j},s,t}$ is as follows: a leaf must acquire the t-th item from stream s_s 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_s 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 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 $\mathcal{L}_{s,t}$. 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 node as $l_{i,j}$ that have been evaluated 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_s. 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. □

The expected cost of a schedule ξ can be computed from Eq. (1) and Proposition 1. We now derive the complexity of carrying out this computation. Recall that L is the total number of

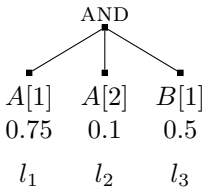


Figure 3: Example *shared* AND tree for which the *read-once* algorithm in [7] is not optimal.

leaves in the tree. Let D be the maximum number of required data items over all streams. More formally, $L = \sum_{i=1}^{N} m_i$ and $D = \max_{1 \leq i \leq N, 1 \leq j \leq m_i} d_{l_{i,j}}$. To compute all the sets $\mathcal{L}_{s,t}$ we need to scan the leaves of each AND node according to schedule ξ while recording the maximum number of items required from each stream. This can be done with complexity $O(L)$. Each set $\mathcal{L}_{s,t}$ 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 $O(L + N^2)$ (because the sets $\mathcal{A}_{i,j}$ take at most N distinct values and each contains at most N elements). Computing all the product of probabilities used in the computation of all the $\mathcal{C}_{l_{i,j},s,t}$ 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 $\mathcal{C}_{l_{i,j},s,t}$ can be computed in $O(N)$, the second in $O(N^2)$, and the third one in $O(1)$. We compute $\mathcal{C}_{l_{i,j},s,t}$ for all the streams required by the leaf $l_{i,j}$ and the maximum number of streams required by a leaf is S. Overall the cost of a schedule can be evaluated with complexity

$$O(LSDN^2).$$

5 AND trees

In this section we focus on AND tree for *shared* queries. We first show that the optimal algorithm for *read-once* queries is no longer optimal for *shared* queries (Section 5.1). We develop an optimal greedy algorithm in the *single-stream* case (Section 5.2). We show that the problem is NP-complete in the *multi-stream* case, for which we propose a heuristic that we show to be close to the optimal for small instances (Section 5.3).

5.1 Is the optimal *read-once* algorithm still optimal?

One valid question is whether the algorithm in [7], which is optimal for *read-once* queries, is still optimal for *shared* queries. It turns out that it is not, and in this section we provide a counter-example. Consider the AND tree depicted in Figure 3 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, the number of data items needed from that stream to evaluate the leaf, and the success probability (p_i). For instance, leaf l_2 requires $d_{l_2}^A = 2$ items from stream A and evaluates to TRUE with probability $p_2 = 0.1$. We assume that retrieving a data item from any stream has unitary cost. There are 6 possible schedules for this tree, each schedule corresponding to one of the 3! orderings of the leaves. The algorithm in [7] sorts the leaves by non-decreasing $d_{l_j}^s c(s)/q_j$, where s is the only stream from which l_j requires data items. 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 for *shared* AND trees.

5.2 The *single-stream* case

In this section we give an optimal algorithm for solving the problem for *shared* AND trees in the *single-stream* case. Like the algorithm in [7] for *read-once* queries, our algorithm is greedy. But it 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 with a preliminary result on the optimal ordering of leaves that use the same stream.

5.2.1 Ordering same-stream leaves

In the example given in Section 5.1 we consider 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 2. *Consider an* AND *tree and a leaf l_i that requires $d^s_{l_i} > 0$ data items from a stream s. In an optimal schedule, l_i is scheduled before any leaf l_j that requires $d^s_{l_j} > d^s_{l_i}$ data items from stream s.*

Proof. This proposition is proven via a simple exchange argument (see B). □

5.2.2 Optimal schedule

Consider an AND tree with L leaves, $l_1, \ldots, l_L$, for S streams, $s_1, \ldots, s_S$. We define $\mathcal{L}_k = \{l_j \mid d^{s_k}_{l_j} > 0\}$, i.e., the set of leaves that require data items from stream s_k. We propose a greedy algorithm, SingleStreamGreedy (Algorithm 1). This algorithm, which is implemented recursively for clarity of presentation, takes as input the $\mathcal{L}_k$ sets, an initially empty schedule ξ, and an array of S integers, *NumItems*, 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 the minimum such ratio is selected (leaf l_{j_0} in the algorithm, which requires $d^{s_{k_0}}_{l_{j_0}}$ data items from stream s_{k_0}). In the last loop of the algorithm, all unscheduled leaves that require $d^{s_{k_0}}_{l_{j_0}}$ or fewer data items from stream s_{k_0} are appended to the schedule in increasing order of the number of required data items.

Theorem 1. SingleStreamGreedy *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 SingleStreamGreedy, ξ_{greedy}, is not optimal. Among

Algorithm 1: SINGLESTREAMGREEDY($\{\mathcal{L}_1, ..., \mathcal{L}_S\}, \xi, NumItems$)

if $\cup_{i=1}^{S} \mathcal{L}_i = \varnothing$ **then return** ξ
$MinRatio \leftarrow +\infty$
for $k = 1$ *to* S **do**
  $Cost \leftarrow 0$
  $Proba \leftarrow 1$
  $Num \leftarrow NumItems[k]$
  for $l_j \in \mathcal{L}_k$ *by non-decreasing* $d_{l_j}^{s_k}$ **do**
    $Cost \leftarrow Cost + Proba \times (d_{l_j}^{s_k} - Num) \times c(k)$
    $Proba \leftarrow Proba \times p_j$
    $Num \leftarrow d_{l_j}^{s_k}$
    $Ratio \leftarrow \frac{Cost}{(1-Proba)}$
    if $Ratio < MinRatio$ **then**
      $MinRatio \leftarrow Ratio$
      $j_0 \leftarrow j$; $k_0 \leftarrow k$
for l_j *in* $\mathcal{L}_{k_0}$ *by non-decreasing* $d_{l_j}^{s_{k_0}}$ **do**
  if $d_{l_j}^{s_{k_0}} \leq d_{l_{j_0}}^{s_{k_0}}$ **then**
    $\xi \leftarrow \xi \cdot l_j$
    $\mathcal{L}_{k_0} \leftarrow \mathcal{L}_{k_0} \setminus \{l_j\}$
$NumItems[k_0] \leftarrow d_{l_{j_0}}^{s_{k_0}}$
return SINGLESTREAMGREEDY($\{\mathcal{L}_1, ..., \mathcal{L}_S\}, \xi, NumItems$)

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 SINGLESTREAMGREEDY 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, $1 \leq i \leq k$, 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 lengthy and technical and the full proof is provided in C.

□

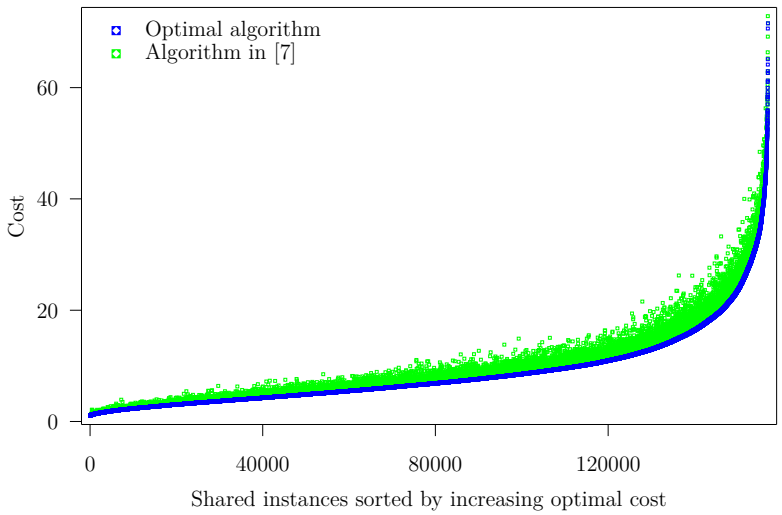


Figure 4: Cost achieved by the algorithm in [7] and that achieved by the optimal SingleStream-Greedy algorithm, for 157,000 random AND tree instances sorted by increasing optimal cost.

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

One may wonder how the optimal algorithm for *read-once* queries [7], which simply sorts the leaves by increasing $d^s_{l_j}c(s)/q_j$, fares for *shared* queries. In other terms, is SingleStream-Greedy 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 $L = 2, \ldots, 20$ 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. These results show that 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 only 11.29% of the instances. We conclude that, for *shared* queries, SingleStreamGreedy provides substantial improvements over the optimal algorithm for *read-once* queries.

5.3 The *multi-stream* case

In this section, we first show the NP-completeness of determining the optimal schedule for a *multi-stream* AND tree. Next we show how to extend the greedy algorithm of the *single-stream* case. While no longer optimal, this extended greedy algorithm is close to the optimal in practice and thus proves useful for designing heuristics.

5.3.1 The *multi-stream* case is NP-complete

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

Theorem 2. AND-Multi-Decision *is NP-complete.*

Proof. The problem is clearly in NP: given a schedule, i.e., an ordering of the leaves, one can compute its expected cost in polynomial time (using the method given in Section 4) and compare it to K. The NP-completeness is obtained by reduction from 2-PARTITION [15]. Let $\mathcal{I}_1$ be an instance from 2-PARTITION: given a set $\{a_1, ..., a_n\}$ and $S = \sum_{i=1}^{n} a_i$, does there exist a subset I such that $\sum_{i \in I} a_i = \frac{S}{2}$? We assume that S is even, otherwise there is no solution. The size of $\mathcal{I}_1$ is $O(n \times \log M)$, where $M = \max_{1 \leq i \leq n}\{a_i\}$. Without loss of generality, we assume that $M \geq 10$. We construct the following instance $\mathcal{I}_2$ of AND-Multi-Decision:

- We consider an AND tree with $n+1$ leaves ℓ_i, $1 \leq i \leq n+1$. The set of streams is $\mathcal{S} = \{A_1, \ldots, A_n, B\}$. The cost of stream $s_i = A_i$ for $i \leq n$ is $c(i) = \frac{1}{2Z}$, where Z is some large constant defined hereafter. The cost of stream $s_{n+1} = B$ is $c(n+1) = C_0$, where $C_0 \approx \frac{1}{2}$ is a constant defined hereafter.
- The first n leaves have a single stream: for $1 \leq i \leq n$, leaf ℓ_i accesses $2a_i$ elements of stream A_i, so that the cost of evaluating leaf ℓ_i (without re-use) is $\frac{a_i}{Z}$. The success probability of leaf ℓ_i is
$$p_i = 1 - \frac{a_i}{Z} - \beta \frac{a_i^2}{Z^2},$$
where $\beta \approx \frac{1}{2}$ is a constant defined hereafter.
- Leaf ℓ_{n+1} accesses all $n+1$ streams: one element of the stream B, and a_i elements of each of the n streams A_i. The cost of evaluating leaf ℓ_{n+1} (assuming no re-use of data items acquired during the evaluation of other leaves) is $C = C_0 + \sum_{i=1}^{n} \frac{a_i}{2Z} = C_0 + \frac{S}{2Z}$. The success probability of leaf ℓ_{n+1} is $p_{n+1} = \varepsilon$. Constant ε is chosen to be very small, see below. Intuitively, C would be the cost of a schedule evaluating leaf ℓ_{n+1} first, and thereby terminating the evaluation, when ε becomes negligible.
- The bound on the expected evaluation cost is $K = C\left(1 - \frac{S^2}{8Z^2}\right) + \frac{1}{9Z^2}$.

To finalize the description of $\mathcal{I}_2$, we define the constants as follows:

- $Z = 10\left((n+1)3^n + n^3\right) M^3$,
- $C_0 = \frac{Z}{2Z-S} - \frac{S}{2Z}$, so that $C = \frac{Z}{2Z-S}$,
- $\beta = \frac{1-C}{2C}$,
- $\varepsilon = \frac{1}{(n+1)^2 90 Z^2}$.

The size of $\mathcal{I}_2$ is polynomial in the size of $\mathcal{I}_1$: the greatest value in $\mathcal{I}_2$ is Z and $\log(Z)$ is linear in $(n + \log M)$. Because Z is very large relatively to $S \leq nM$, we do have that C, C_0, and β are all close to $\frac{1}{2}$. We only use that these constants are all nonnegative, and that $\beta \leq 1$ and $C \leq 1$, in the following derivation, where we bound the expected cost of an arbitrary evaluation of the AND tree. Then, using this derivation, we prove that $\mathcal{I}_1$ has a solution I if and only if $\mathcal{I}_2$ does.

Let us start with the cost of an arbitrary evaluation of the AND tree. In such an evaluation, we evaluate some (possibly none) of the first n leaves before evaluating leaf ℓ_{n+1}. Then, because

ε is small, we can compute an approximation of the cost as follows: we assume that the schedule terminates after leaf ℓ_{n+1}, because its success probability is close to 0. We will bound the difference between this approximation and the actual cost later on.

Let $I = \{\ell_{\sigma(1)}, \ell_{\sigma(2)}, \ldots, \ell_{\sigma(k)}\}$ be the subset, of cardinal k, of leaves that are evaluated, in that order, before leaf ℓ_{n+1}. Let $\overline{Cost}$ be the approximated cost of the schedule (terminating after completion of leaf ℓ_{n+1}). To simplify notations, we let $x_i = a_{\sigma(i)}$ and $r_i = p_{\sigma(i)}$ for $1 \leq i \leq k$. By definition:

$$\overline{Cost} = \sum_{i=1}^{k} \frac{x_i}{Z} \prod_{1 \leq j < i} r_j + \left(C - \sum_{i=1}^{k} \frac{x_i}{2Z}\right) \prod_{1 \leq j \leq k} r_j.$$

Note that the cost of leaf ℓ_{n+1} has been reduced from its original value, due to the sharing of the streams whose index is in I. To evaluate $\overline{Cost}$, we start by approximating

$$\prod_{1 \leq j < i} r_j = \prod_{1 \leq j < i} \left(1 - \frac{x_j}{Z} - \beta \frac{x_j^2}{Z^2}\right).$$

Let

$$F_i = 1 - \sum_{j=1}^{i-1} \frac{x_j}{Z} - \beta \sum_{j=1}^{i-1} \frac{x_j^2}{Z^2} + \sum_{1 \leq j_1 < j_2 < i} \frac{x_{j_1} x_{j_2}}{Z^2}.$$

We have

$$\left| \left(\prod_{1 \leq j < i} r_j \right) - F_i \right| \leq \frac{3^n M^3}{Z^3}. \tag{2}$$

To see this, we have kept in F_i all terms of the product $\prod_{1 \leq j < i} r_j$ whose denominators include a factor strictly inferior to Z^3. The other terms of the product are bounded (in absolute value) by M^3/Z^3, because $\beta \leq 1$, $x_j \leq M$, and $M \leq Z$. There are at most $3^{i-1} \leq 3^n$ such terms. Hence the desired bound in Equation (2). Letting

$$G = \sum_{i=1}^{k} \frac{x_i}{Z} - \sum_{1 \leq j_1 < j_2 \leq k} \frac{x_{j_1} x_{j_2}}{Z^2},$$

we prove similarly that

$$\left| \left(\sum_{i=1}^{k} \frac{x_i}{Z} \prod_{1 \leq j < i} r_j \right) - G \right| \leq \frac{n 3^n M^3}{Z^3}. \tag{3}$$

Indeed, there are $k \leq n$ terms in the sum, each of them being bounded as before. We deduce from Equations (2) and (3), using $C \leq 1$, that

$$\left| \overline{Cost} - \left(G + (C - \sum_{i=1}^{k} \frac{x_i}{2Z}) F_{k+1} \right) \right| \leq \frac{(n+1) 3^n M^3}{Z^3}. \tag{4}$$

Now, we aim at simplifying $H = G + (C - \sum_{i=1}^{k} \frac{x_i}{2Z}) F_{k+1}$ by dropping terms whose denominator is Z^3. We have

$$H = \sum_{i=1}^{k} \frac{x_i}{Z} - \sum_{1 \leq j_1 < j_2 \leq k} \frac{x_{j_1} x_{j_2}}{Z^2} + \left(C - \sum_{i=1}^{k} \frac{x_i}{2Z}\right) \left(1 - \sum_{j=1}^{k} \frac{x_j}{Z} - \beta \sum_{j=1}^{k} \frac{x_j^2}{Z^2} + \sum_{1 \leq j_1 < j_2 \leq k} \frac{x_{j_1} x_{j_2}}{Z^2}\right).$$

Defining

$$\tilde{H} = C + \frac{1-2C}{2Z}\sum_{i=1}^{k} x_i + \frac{1}{2Z^2}\left(\sum_{i=1}^{k} x_i\right)^2 + \frac{C-1}{Z^2}\sum_{1\le j_1<j_2\le k} x_{j_1}x_{j_2} - \frac{\beta C}{Z^2}\sum_{i=1}^{k} x_i^2,$$

we derive (using $\beta \le 1$) that:

$$\left|H - \tilde{H}\right| \le \left|\frac{1}{2Z^3}\left(\sum_{i=1}^{k} x_i\right)\left(\sum_{1\le j_1<j_2\le k} x_{j_1}x_{j_2} + \sum_{i=1}^{k} x_i^2\right)\right|.$$

Hence,

$$\left|H - \tilde{H}\right| \le \frac{n^3M^3}{Z^3}. \tag{5}$$

Developing $(\sum_{i=1}^{k} x_i)^2 = \sum_{i=1}^{k} x_i^2 + 2\sum_{1\le j_1<j_2\le k} x_{j_1}x_{j_2}$ in $\tilde{H}$, we obtain

$$\tilde{H} = C + \frac{1-2C}{2Z}\sum_{i=1}^{k} x_i + \frac{C}{Z^2}\sum_{1\le j_1<j_2\le k} x_{j_1}x_{j_2} + \frac{1-2\beta C}{2Z^2}\sum_{i=1}^{k} x_i^2.$$

We have chosen the constants C and β so that $\tilde{H}$ can be reduced to

$$\tilde{H} = C + \frac{C}{2Z^2}\left(\left(\frac{S}{2} - \sum_{i=1}^{k} x_i\right)^2 - \frac{S^2}{4}\right). \tag{6}$$

Indeed, we have $\frac{1-2C}{2Z} = \frac{-SC}{2Z^2}$, and $C = 1 - 2C\beta$. Altogether, we derive from Equations (4) to (6) that

$$\left|\overline{Cost} - C\left(1 - \frac{S^2}{8Z^2}\right) - \frac{C}{2Z^2}\left(\frac{S}{2} - \sum_{i=1}^{k} x_i\right)^2\right| \le \frac{\left((n+1)3^n + n^3\right)M^3}{Z^3} = \frac{1}{10Z^2}. \tag{7}$$

Finally, we coarsely bound the difference between the actual cost *Cost* of the schedule and the approximated cost $\overline{Cost}$. The actual probability of evaluating some other leaves after leaf ℓ_{n+1} is ε, there are at most n such leaves, whose individual cost does not exceed $\frac{M}{2Z}$. We get a difference bounded by $n\frac{M}{2Z}\varepsilon$, from which we derive

$$\left|Cost - \overline{Cost}\right| \le n\frac{M}{2Z}\varepsilon \le (n+1)^2\varepsilon = \frac{1}{90Z^2}. \tag{8}$$

We could easily tighten the bound in Equation (8), but we will keep the same notations to derive a similar bound in the proof of Theorem 4.

Combining Equations (7) and (8), we finally derive that

$$\left|Cost - C\left(1 - \frac{S^2}{8Z^2}\right) - \frac{C}{2Z^2}\left(\frac{S}{2} - \sum_{i=1}^{k} x_i\right)^2\right| \le \frac{1}{9Z^2}. \tag{9}$$

We now prove that $\mathcal{I}_1$ has a solution I if and only if $\mathcal{I}_2$ does. Suppose first that $\mathcal{I}_1$ has a solution I: $\sum_{i\in I} a_i = \frac{S}{2}$. We evaluate the leaves whose indices are in I before evaluating leaf

ℓ_{n+1}, followed by the remaining leaves in any order. Let *Cost* be the cost of this evaluation. From Equation (9), we have

$$\left| Cost - C\left(1 - \frac{S^2}{8Z^2}\right)\right| \leq \frac{1}{9Z^2}.$$

Hence, $Cost \leq C\left(1 - \frac{S^2}{8Z^2}\right) + \frac{1}{9Z^2} = K$, thereby providing a solution to $\mathcal{I}_2$.

Suppose now that $\mathcal{I}_2$ has a solution whose cost is $Cost \leq K$, and let I denote the (index) set of leaves that are evaluated before leaf ℓ_{n+1}. If (by contradiction) we have $\sum_{i\in I} a_i \neq \frac{S}{2}$, then $\left(\frac{S}{2} - \sum_{i=1}^{k} x_i\right)^2 \geq 1$, and Equation (9) shows that

$$Cost \geq C\left(1 - \frac{S^2}{8Z^2}\right) + \frac{C}{2Z^2} - \frac{1}{9Z^2} = K + \frac{9C-4}{9Z^2}.$$

Since $9C - 4 = \frac{Z+4S}{2Z-S}$, $9C - 4 > 0$. Then, $Cost > K$ and we obtain a contradiction. Therefore $\sum_{i\in I} a_i = \frac{S}{2}$, and $\mathcal{I}_1$ has a solution, which concludes the proof.

□

5.3.2 Greedy heuristic for the *multi-stream* case

Since AND-Multi-Decision is NP-complete we propose a greedy scheduling heuristics, MultiStreamGreedy (Algorithm 2), which extends the ideas of SingleStreamGreedy to the *multi-stream* case.

SingleStreamGreedy computes a schedule by concatenating sequences of leaves. Each such sequence consists of leaves that all require data items from the same stream. These leaves are ordered by non-decreasing number of required data items. We extend this approach to the *multi-stream* case thanks to a notion of *dominance*. We say that leaf l_i dominates leaf l_j if for each stream s leaf l_i requires at least as many data items from s as l_j ($d^s_{l_i} \geq d^s_{l_j}$). MultiStreamGreedy considers all sequences of yet-to-be-scheduled leaves such that the $(i+1)$-th leaf in the sequence dominates the i-th leaf. Like SingleStreamGreedy, MultiStreamGreedy picks the sequence that has the lowest ratio of cost to probability of failure.

Consider an AND tree with a set of leaves $\mathcal{L} = l_1, \ldots, l_L$. MultiStreamGreedy takes $\mathcal{L}$ as input and returns a schedule, ξ, and its expected cost. While there remain leaves to schedule (while loop at line 6), MultiStreamGreedy computes all dominance relationships among the yet to be scheduled leaves via a call to DirectDomination (Algorithm 6 in D). DirectDomination returns the set of source leaves, i.e., leaves that dominate no other leaves (boolean array *Source*), and the set of dominance relationships (boolean array *Dominates*). These relationships are direct, i.e., not including transitive dominances. For each source leaf l_i, MultiStreamGreedy then examines all possible sequences starting with l_i (for loop at line 11). This is done via a call to the recursive GreedyKernel function (Algorithm 3), which computes the sequence that starts with l_i that provides the best extension to the current schedule. MultiStreamGreedy then selects the best extension among all these extensions for all source leaves (lines 14-18).

The complexity of GreedyKernel is $O(2^L)$. This is because GreedyKernel explores all paths starting from a given source leaf in the dominance relationship graph (in a directed acyclic graph with n vertices there are at most $O(2^n)$ paths). MultiStreamGreedy has complexity $O(L^2 2^L)$. This is because at each step MultiStreamGreedy schedules at least one leaf and calls GreedyKernel for each unscheduled source leaf. In spite of its exponential worst-case complexity, for the problem instances used in our experimental evaluations we are able to execute

GreedyKernel in at most 0.03 sec (for a AND node with 40 leaves) on one core of an 2.1 GHz AMD Opteron processor.

Figure 5 shows results for a set of randomly generated AND trees. Instances are generated using the same method as that described in Section 5.2.2. For a given number of leaves $L = 2, \ldots, 10$ 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 81,000 random trees. The number of streams referenced by each leaf is sampled from a uniform distribution over the interval $[1, 5]$, and each such stream is sampled uniformly from the set of streams. For each tree we compute the cost achieved by MultiStreamGreedy and that achieved by a high-complexity exhaustive search for the optimal schedule (of complexity $O(L!)$). Figure 5 plots these costs for all instances, sorted by increasing optimal cost. The average, resp. maximum, relative difference between the results of MultiStreamGreedy and the results of the optimal algorithm is 0.60%, resp. 28.53%. The relative difference is larger than 5% for only 3.73% of the instances, and the two algorithms lead to the same cost for 76.75% of the instances. We conclude that MultiStreamGreedy is likely to provide close-to-optimal schedules in *multi-stream* case for *shared* queries.

Algorithm 2: MultiStreamGreedy($\mathcal{L}$)

```
1  for l_i ∈ L do
2      NotYetScheduled[l_i] ← TRUE
3  NumNotYetScheduled ← |L|
4  ScheduleCost ← 0
5  ξ ← ∅
6  while (NumNotYetScheduled > 0) do
7      (Source, Dominates) = DirectDomination(L, NotYetScheduled)
8      MinRatio ← +∞
9      BestExt ← ∅
10     CostBestExtension ← +∞
11     for l_i ∈ L do
12         if (NotYetScheduled[l_i] and Source[l_i]) then
13             (Cost, Extension, Proba) =
                   GreedyKernel(L, ξ, ScheduleCost, 1, l_i, Dominates)
14             Ratio ← (Cost − ScheduleCost)/(1 − Proba)
15             if (Ratio < MinRatio) then
16                 MinRatio ← Ratio
17                 BestExt ← Extension
18                 CostBestExtension ← Cost
19     ξ ← ξ · BestExt
20     ScheduleCost ← CostBestExtension
21     for l_i ∈ BestExt do
22         NotYetScheduled[l_i] ← FALSE
23         NumNotYetScheduled ← NumNotYetScheduled − 1
24 return (ξ, ScheduleCost)
```

Algorithm 3: GreedyKernel($\mathcal{L}, \xi, BaseCost, BaseProba, Leaf, Dominates$)

```
1  CostBestSol ← Cost(ξ · Leaf)
2  BestExt ← Leaf
3  ProbaBestExt ← p_Leaf
4  BestRatio ← (CostBestSol − BaseCost)/(1 − BaseProba × p_Leaf)
5  for l_j ∈ L do
6      if Dominates[l_j][Leaf] then
7          (Cost, Ext, Proba) ←
           GreedyKernel(L, ξ · Leaf, BaseCost, BaseProba × p_Leaf, l_j, Dominates)
8          Ratio ← (Cost − BaseCost)/(1 − BaseProba × p_Leaf × Proba)
9          if (Ratio < BestRatio) then
10             BestRatio ← Ratio
11             CostBestSol ← Cost
12             ProbaBestExt ← p_Leaf × Proba
13             BestExt ← Leaf · Ext
14 return (CostBestSol, BestExt, ProbaBestExt)
```

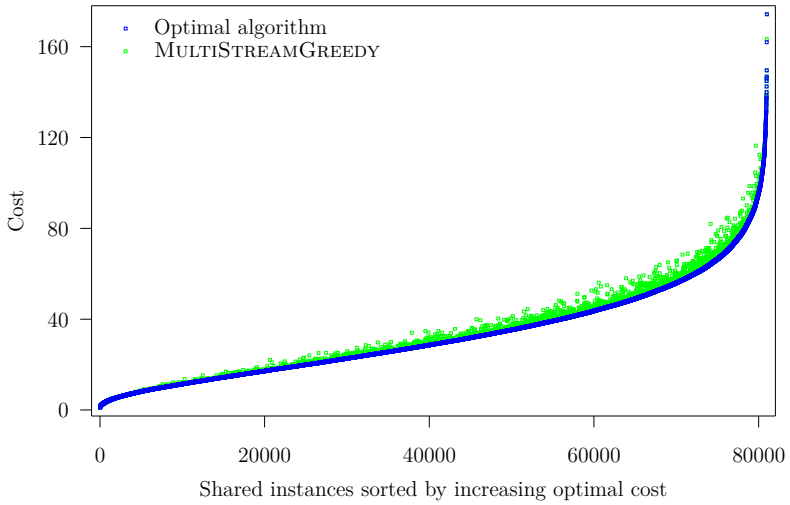


Figure 5: Cost achieved by MultiStreamGreedy and that achieved by the optimal algorithm, shown for 81,000 random AND tree instances sorted by increasing optimal cost.

6 DNF trees

In this section we focus on DNF trees for *shared* queries. We first show that, as for *read-once* queries, there is an optimal schedule that is depth-first (Section 6.1). This result holds in the *multi-stream* case, and thus in the less general *single-stream* case. We then show that the problem is NP-complete in the *single-stream* case (Section 6.2), and thus also NP-complete in the more general *multi-stream* case. We then propose and evaluate several heuristics (Section 6.3).

6.1 Dominance of depth-first schedules

Theorem 3. *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. We use the same notations as in Section 4. Without loss of generality we assume that the AND nodes, $\text{AND}_1, \ldots, \text{AND}_N$, are numbered in the order of their completion in ξ. Thus, according to ξ, AND_1 is the first AND node with all its leaves evaluated. We denote by K the number (possibly zero) of AND nodes that are processed one by one and entirely at the beginning of the query processing according to ξ. Therefore, if ξ evaluates a leaf $l_{i,j}$, with $i \neq 1$, in the m_1 first steps, then $K = 0$. Finally, we assume that the leaves of each 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 an optimal schedule with $K = N$. Let ξ be an optimal schedule that maximizes K. By definition of K and by the hypothesis on the numbering of the AND nodes, schedule ξ evaluates some leaves of the AND nodes AND_{K+2}, ..., AND_N before it evaluates the last leaf of AND_{K+1}. Let $\hat{\mathcal{L}}$ denote the set of these leaves. We now define a new schedule ξ' that starts by executing at least $K+1$ AND nodes one by one:

- ξ' starts by evaluating the first K 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 AND_{K+1} in the same order as in ξ (but not at the same steps);
- ξ' then evaluates the leaves in $\hat{\mathcal{L}}$ 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 any leaf of AND_{K+1} and not needed by any leaf in $\hat{\mathcal{L}}$. 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 AND_{K+1}. The only way in which a leaf that is evaluated in ξ would not be evaluated in ξ' is if AND_{K+1} evaluates to TRUE. Since at least one leaf of AND_{K+1} uses d, for AND_{K+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 $\hat{\mathcal{L}}$ but not needed by any leaf of AND_{K+1}. ξ and ξ' define the same ordering on the leaves in $\hat{\mathcal{L}}$. For each AND node AND_i, with $K+2 \leq i \leq N$, there is at most one leaf in $\text{AND}_i \cap \hat{\mathcal{L}}$ 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:

- AND_1, ..., AND_K all evaluate to FALSE;
- None of the evaluated leaves of AND_1, ..., AND_K 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 AND_1, ..., AND_K 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 AND_1, ..., AND_K. Finally, let $\mathcal{R}$ be the probability that one of the leaves evaluated with ξ after $l_{K+1,m_{K+1}}$ acquires d, knowing that no leaves of AND_1, ..., AND_K or in $\hat{\mathcal{L}}$ acquires it. Then, with ξ, the probability p that d is acquired is:

$$p = \mathcal{D} + \mathcal{P}\left(1 - \prod_{l_{i,j} \in \mathcal{F}} \left(1 - \prod_{k=1}^{j-1} p_{i,k}\right)\right) + \mathcal{R} \tag{10}$$

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 (10) 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 AND_1, ..., AND_K, and AND_{K+1} all evaluate to FALSE;
- None of the evaluated leaves of the AND nodes AND_1, ..., AND_K 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:

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

Comparing this equation with Equation 10, 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 $(K+1)$ AND nodes one by one, we obtain a contradiction with the maximality assumption on K, which concludes the proof. □

6.2 The *single-stream* case is NP-complete

For *read-once* queries an optimal algorithm for DNF trees is built on top of the optimal algorithm for AND trees in [8]. The same approach cannot be used for *shared* queries (i.e., reusing SingleStreamGreedy). This can be shown by a simple counter-example (see F). 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 SingleStreamGreedy for that AND node. In fact, we show that finding an optimal schedule for evaluating a DNF tree is NP-complete.

Definition 2 (DNF-Single-Decision)**.** *Given a* single-stream DNF *tree and a cost bound K, is there a schedule whose expected cost does not exceed K?*

Theorem 4. DNF-Single-Decision *is NP-complete.*

Proof. The problem is clearly in NP since the expected cost of a schedule can be computed in polynomial time (see Section 4) and compared to K. NP-completeness is obtained via a proof very similar to that of Theorem 2. We build the following instance of $\mathcal{I}_2$ of DNF-Decision:

- We consider a DNF tree with $N = n + 1$ AND nodes AND_i, $1 \leq i \leq n+1$, and a total of $L = 2n + 1$ leaves.
- The set of streams is $\mathcal{S} = \{A_1, \ldots, A_n, B\}$. The cost of stream $s_i = A_i$ for $i \leq n$ is $c(i) = \frac{1}{2Z}$. The cost of stream $s_{n+1} = B$ is $c(n+1) = C_0$

- Each AND_i node, where $i \leq n$, has a single leaf that has success probability

$$p_{i,1} = \frac{a_i}{Z} + \beta \frac{a_i^2}{Z^2},$$

  and which requires $d_{l_{i,1}}^{A_i} = 2a_i$ elements of stream $s_i = A_i$.
- The last AND node AND_{n+1} has $m_{n+1} = n + 1$ leaves specified as follows:
  - Each leaf $l_{n+1,i}$, where $i \leq n$, has success probability $p_{n+1,i} = 1 - \varepsilon$ and requires $d_{l_{n+1,i}}^{A_i} = a_i$ elements of stream $s_i = A_i$.
  - The last leaf $l_{n+1,n+1}$ has success probability $p_{n+1,n+1} = 1-\varepsilon$ and requires $d_{l_{n+1,n+1}}^{B} = 1$ element of stream $s_{n+1} = B$ (at cost $c(n+1) = C_0$).
  - All constants C_0, C, K, Z, β, and ε are those in the proof of Theorem 2.

In an arbitrary evaluation of the DNF tree, we evaluate some (possibly none) of the first n AND nodes before starting to evaluate node AND_{n+1}. The intuition is to schedule a good subset of the first n AND nodes before scheduling the last AND node, in order to achieve a trade-off: the last AND leads to success with high probability but starting by executing it has a high cost. The reduction is then very similar to that of Theorem 2. See the full proof in E.

□

It is interesting to point out that in the above proof instance $\mathcal{I}_2$ is constructed so that the ordering of the leaves inside each AND node has no importance. In fact, only the last AND node has more than one leaf, and because its leaves have all high success probability, their ordering does not matter. This shows that the combinatorial difficulty of the DNF-Decision problem already lies in deciding the ordering of the AND nodes. However, even if the optimal order of the AND nodes is given, an optimal schedule cannot be computed by simply using SingleStreamGreedy (which is optimal for a single AND node) for scheduling the leaves of each AND node. See a counter-example in F.

6.3 Heuristics and Evaluation Results

Given the NP-completeness result in the previous section we propose several heuristics. Most of these heuristics apply to both the *single-stream* and the *multi-stream* cases, one heuristic applies only to the *single-stream* case, and another applies only to the *multi-stream* case, as described in the following sections.

6.3.1 Heuristics common to the *single-stream* and *multi-stream* cases

We propose two categories of heuristics, which we term leaf-ordered and AND-ordered. *Leaf-ordered heuristics* simply sort the leaves according to 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, non-increasing q (prioritizes leaves with high chances of shortcutting the evaluation of an AND node);
- Leaf-ordered, non-decreasing $\mathcal{C}$ (prioritizes leaves with low costs);
- Leaf-ordered, non-decreasing $\mathcal{C}/q$ (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 non-increasing $\mathcal{C}$) 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 3). The heuristics thus proceed in two phases: (i) compute a schedule for the leaves of each AND node independently, ignoring the other AND nodes, using SingleStream-Greedy (optimal) in the *single-stream* case and MultiStreamGreedy (sub-optimal) in the *multi-stream* case; (ii) pick an order of the AND nodes and concatenate their individual leaf schedules. Given the individual schedule of each AND node computed in the first phase, we can compute the (expected) cost and the probability of success of that AND node (using the method in Section 4). In the second phase, the 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, leading to three heuristics:

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

6.3.2 A greedy stream-ordered heuristic for the *single-stream* case

For the *single-stream* case we propose a heuristic that orders the streams for data acquisition. The idea of this heuristic was proposed in [5], and to the best of our knowledge it is the only previously proposed heuristic for solving the PAOTR problem for *shared* DNF query trees.

The *stream-ordered* heuristic proceeds 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. For each stream s_s the heuristic computes a metric, $R(s_s)$, defined as follows:

$$R(s_s) = \frac{\sum_{i,j|d^{s_s}_{l_{i,j}}>0} q_{i,j} n_{i,j}}{\max_{i,j|d^{s}_{l_{i,j}}>0} d^{s_s}_{l_{i,j}} c(s_s)},$$

where $n_{i,j}$ is the number of leaves 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_s. The denominator is the maximum data element acquisition cost over all the leaves that use stream s_s. The heuristic orders the streams by non-decreasing 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 [5] 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 stream s are scheduled in decreasing $d^{s_s}_{l_{i,j}}$ order. However, Proposition 2 holds for DNF trees, showing that it is always better to schedule these leaves in *increasing* $d^{s_s}_{l_{i,j}}$ 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 [5] in the vast majority of the cases, with all remaining cases being ties.

6.3.3 A dynamic programming heuristic for the *multi-stream* case

When faced with an intractable problem, one option is find a related problem for which an optimal solution can be computed. In this view we propose a dynamic programming heuristic,

MultiStreamDP (Algorithm 4), which computes not an optimal order of the leaves, but an optimal order of the individual data item retrievals. The hope is that these two objectives are sufficiently related that the optimal data item retrieval order induces a good leaf schedule.

MultiStreamDP builds a data item retrieval order as follows. Let $Max[s]$ be the maximum number of data items from stream s_s required by any leaf. The algorithm constructs an S-dimensional array DPCost. DPCost$[n_1, \ldots, n_S]$ denotes the expectation of the cost to acquire all the potentially needed data items knowing that n_i data items have already been acquired from stream s_i for all $i = 1, \ldots, S$. The goal is to compute DPCost$[0, \ldots, 0]$. Algorithm 4 first computes $Max[s]$ for all s (lines 1-6). It then iteratively computes DPCost values by non-increasing total number of already acquired data item starting with DPCost$[Max[1], \ldots, Max[S]] = 0$ (lines 7-10). Each computation is accomplish by a call to DPK (Algorithm 5).

DPK begins (lines 1-13) by computing, given the already acquired data items, which leaves are already evaluated, which AND nodes are already fully evaluated, and which streams have data items required by AND nodes that are not fully evaluated. From lines 20 to 29, the algorithm computes for each stream s_s the expected cost of retrieving the next data item from s_s. This computation relies on the probability that all fully evaluated AND nodes evaluate to FALSE (which is computed at lines 14-17). It also relies on the probability that all non-fully evaluated AND nodes requiring at least one data item from s_s so far evaluate to FALSE (which is computed at lines 22-24). The algorithm computes the lowest expected cost when the next acquired data item is from stream s_s (line 26). The desired DPCost value is the lowest such cost.

DPK has complexity $O(SL)$. Let $D = \max_{i \in \{1,\ldots,S\}} Max[i]$. MultiStreamDP places $(D+1)^S$ calls to DPK, for an overall complexity of $O(D^S LS)$. This complexity is exponential in the number of streams, which may preclude the use of MultiStreamDP in practice for instance with more than a few streams.

From the output of MultiStreamDP we must construct a leaf schedule. Based on the data item retrieval order we compute a completion order of the AND nodes, and an evaluation order of the leaves within each AND node. We then construct a depth-first schedule according to these orders.

Algorithm 4: MultiStreamDP $(\mathcal{T})$

```
1  for s = 1 to S do
2      Max[s] ← 0
3      for c = 1 to N do
4          for i = 1 to m_c do
5              if Max[s] < d^{s_s}_{l_{c,i}} then
6                  Max[s] ← d^{s_s}_{l_{c,i}}
7  DPCost[Max[1], ..., Max[S]] ← 0
8  for step = (Σ_{i=1}^{S} Max[i]) − 1 down to 1 do
9      foreach (n_1, ..., n_S) such that Σ_{i=1}^{S} n_i = step, with n_i ≤ Max[i] for each i do
10         DPK(T, DPCost, n_1, ..., n_S)
```

Algorithm 5: DPK($\mathcal{T}$, DPCost, $n_1, ..., n_S$)

```
1  for c = 1 to N do
2      AndCompleted[c] ← True
3      ProbaAndTrue[c] ← 1
4      for s = 1 to S do
5          AndNeedStream[c][s] ← False
6      for i = 1 to m_c do
7          LeafCompleted[i] ← True
8          for s = 1 to S do
9              if n_s < d^s_{l_{c,i}} then
10                 AndCompleted[c] ← False
11                 LeafCompleted[i] ← False
12                 AndNeedStream[c][s] ← True
13         if LeafCompleted[i] then ProbaAndTrue[c] ← ProbaAndTrue[c] × p_{c,i}
14 ProbaAllCompletedAndsFalse ← 1
15 for c = 1 to N do
16     if AndCompleted[c] then
17         ProbaAllCompletedAndsFalse ←
           ProbaAllCompletedAndsFalse × (1 − ProbaAndTrue[c])
18 DPCost[n_1, ..., n_S] ← +∞
19 NextStream[n_1, ..., n_S] ← 0
20 for s = 1 to S do
21     ProbaAllNeedingAndsFalse ← 1
22     for c = 1 to N do
23         if (not AndCompleted[c]) and AndNeedStream[c][s] then
24             ProbaAllNeedingAndsFalse ←
               ProbaAllNeedingAndsFalse × (1 − ProbaAndTrue[c])
25     ProbaStreamRead ←
       ProbaAllCompletedAndsFalse × (1 − ProbaAllNeedingAndsFalse)
26     Cost ← ProbaStreamRead × c(s_s) + DPCost[n_1, ..., n_{s−1}, n_s + 1, n_{s+1}, ..., n_S]
27     if Cost < DPCost[n_1, ..., n_S] then
28         DPCost[n_1, ..., n_S] ← Cost
29         NextStream[n_1, ..., n_S] ← s
```

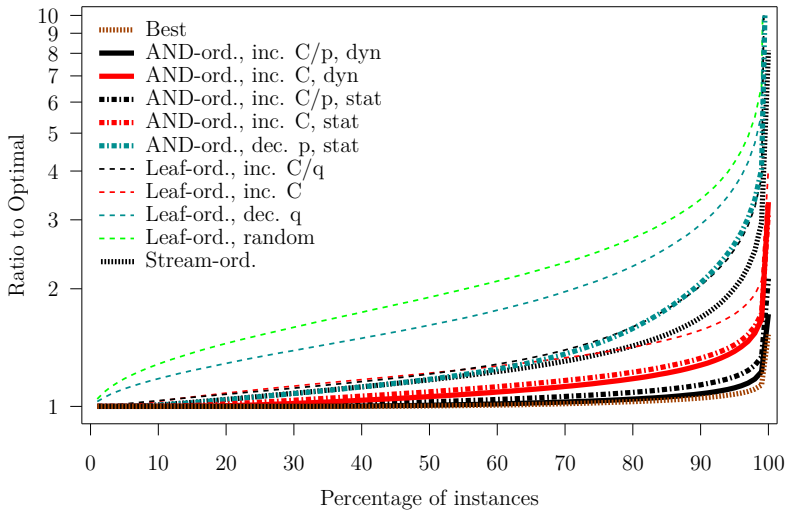


Figure 6: Ratio to optimal vs. fraction of the instances for which a smaller ratio is achieved, computed over 17,100 random "small" DNF tree instances in the *single-stream* case.

6.3.4 Evaluation results for the *single-stream* case

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, thanks to Theorem 3, it only needs to search over all possible depth-first schedules. Instances are generated using the same method as that described in Section 5.2.2 for generating AND tree instances in the *single-stream* case. We generate DNF trees with $N = 2, \ldots, 9$ AND nodes and up to at most 20 leaves and 8 leaves per AND, generating 1,000 random instances for each configuration, for a total of 17,100 instances. For each instance we compute the ratio between the cost achieved by each heuristic and the optimal cost.

Figure 6 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. These results include a curve for a heuristic called "Best." This heuristic runs all other heuristics and returns the generated schedule that achieved the lowest expected cost.

The trends in Figure 6 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. 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 results than the best AND-ordered heuristics. Overall, the most effective heuristic is to sort the AND's by increasing $\mathcal{C}/p$, and it is in fact optimal for 35.46% of the problem instances in Figure 6.

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

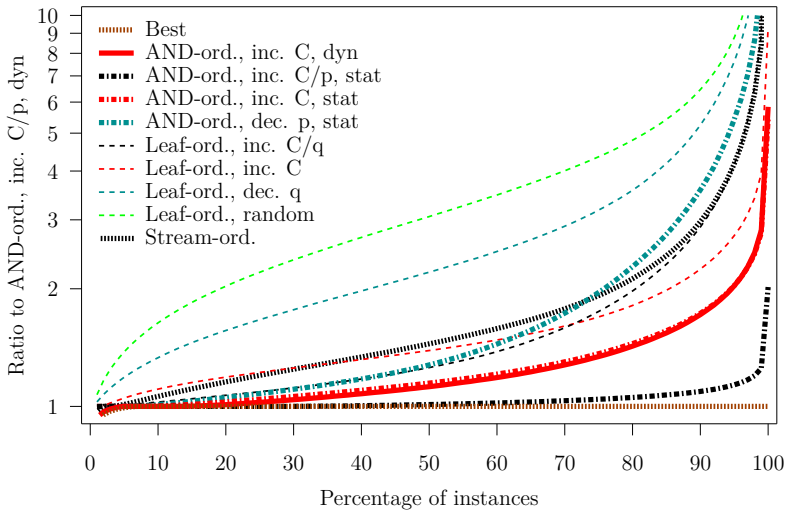


Figure 7: Ratio to Best vs. fraction of the instances for which a smaller ratio is achieved, computed over 32,400 random "large" DNF tree instances in the *single-stream* case.

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 Best heuristic. Results are shown in Figure 7. 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 92.14%, resp. 74.36%, of the problem instances reported in Figure 7, resp. Figure 6. It runs in at most 9 seconds on one core of an 2.1 GHz AMD Opteron processor when processing a tree with 10 AND nodes with each 20 leaves.

6.3.5 Evaluation results for the *multi-stream* case

As in the previous section, we first evaluate our heuristics on a set of "small" instances for which we can compute optimal schedules using an exponential-time exhaustive search (which is feasible because, due to Theorem 3, it only needs to search over all possible depth-first schedules). Instances are generated using the same method as that described in Section 5.3.2 for generating AND tree instances in the *multi-stream* case. We generate DNF trees with $N = 2, \ldots, 8$ AND nodes and up to at most 16 leaves in total and 7 leaves per AND, generating 100 random instances for each configuration, for a total of 16,200 instances. Results are shown in Figure 8, and exhibit clear trends that are similar to those seen in the *single-stream* case. The most effective heuristic is AND-ordered by increasing $\mathcal{C}/p$, dynamic version. This heuristic leads to the optimal solution in 48.98% of the instances, and is the best heuristic in 79.96% of the instances.

These instances are still too large to run the dynamic programming MultiStreamDP heuristic (described in Section 6.3.3) due to its high computational complexity and to its memory requirements. To evaluate this heuristic we generate "very small" instances with $N = 2, \ldots, 5$ AND nodes and up to at most 10 leaves in total and 5 leaves per AND, with a sharing ratio $\rho = 3/2, 2, 3, 4, 5$, or 10 (hence, we only consider the 6 largest of the ratios used in all the other simulations). The number of streams referenced by each leaf is sampled from a uniform

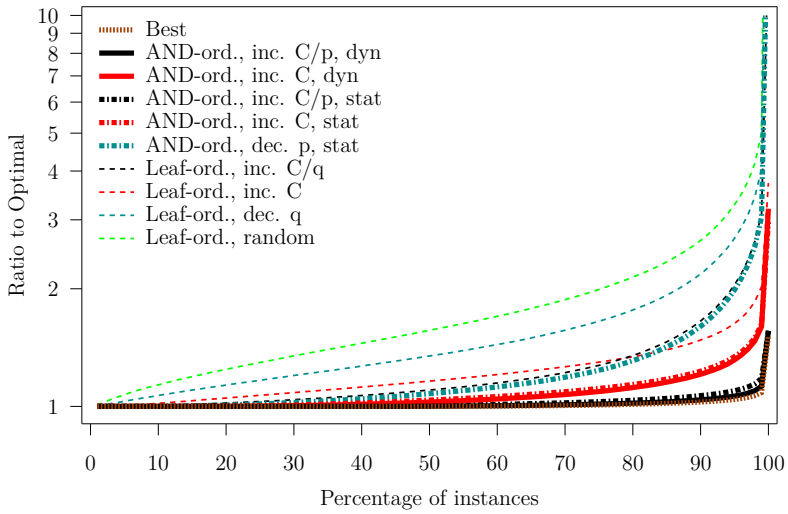


Figure 8: Ratio to optimal vs. fraction of the instances for which a smaller ratio is achieved, computed over 16,200 random "small" DNF tree instances in the *multi-stream* case.

distribution over the interval $[1, 3]$ (rather than $[1, 5]$ as in all other *multi-stream* simulations). We generate 100 random instances for each configuration, for a total of 4,800 instances. Results are shown in Figure 9. Disappointingly, MultiStreamDP achieves results comparable to the best Leaf-ordered heuristic and poorer than all AND-ordered heuristics except the AND-ordered heuristic with decreasing probability of success (p). We conclude that in spite of being optimal for deciding on a stream order, MultiStreamDP leads to poor results for solving the original problem (and is computationally expensive).

We also evaluate our 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. As in the previous section, we compute ratios to the "Best" heuristic since we cannot tractably compute the optimal cost. Results are shown in Figure 10. Essentially, all the observations made on the results for small instances hold. The AND-ordered by increasing $\mathcal{C}/p$, dynamic version, heuristic is the best heuristic in 92.16% of the instances.

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 heuristic in 79.5%, resp. 92.5%, of the cases reported in Figure 8, resp. Figure 10. On one core of an 2.1 GHz AMD Opteron processor, it runs in at most 8 sec when processing trees with 9 AND nodes with each 15 leaves, and in less than 35 sec when processing tress with 10 AND nodes with each 20 leaves.

7 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 [8] in which a single leaf may reference multiple data streams and a single data stream may be referenced by multiple leaves. We have given an optimal algorithm in the case of AND trees in the *single-stream* case. We have shown that the problem is NP-complete for AND trees in the *multi-stream* case and for DNF trees in the *single-stream* case. However, we have shown that there is an optimal leaf evaluation order

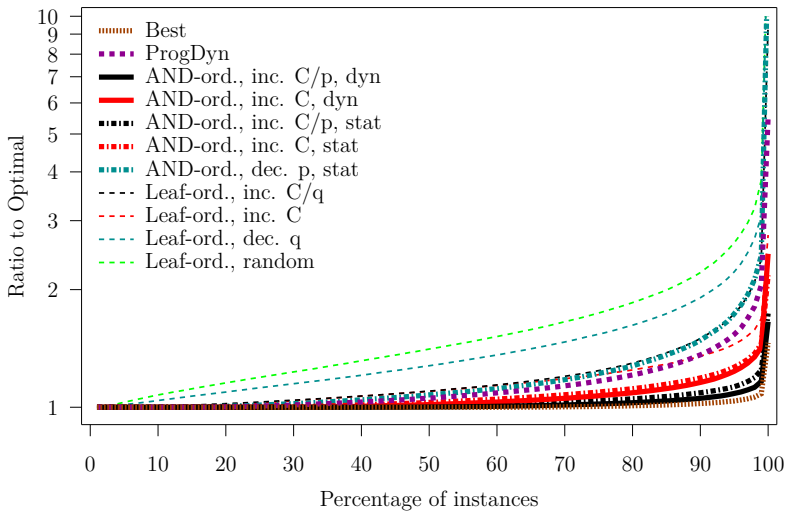


Figure 9: Ratio to optimal vs. fraction of the instances for which a smaller ratio is achieved, computed over 4,800 random "very small" DNF tree instances in the *multi-stream* case.

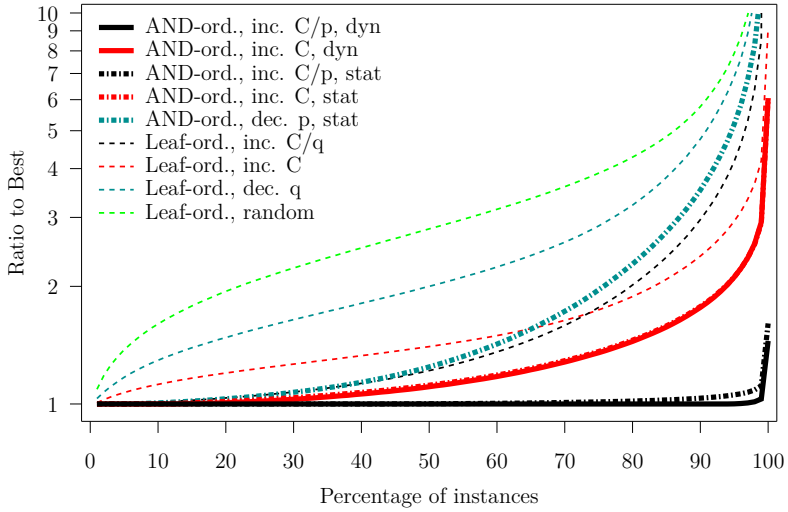


Figure 10: Ratio to the Best heuristic vs. fraction of the instances for which a smaller ratio is achieved, computed over 32,400 random "large" DNF tree instances in the *multi-stream* case.

that corresponds to a depth-first traversal. This observation provides inspiration for designing heuristics that produce depth-first traversals. Numerical results obtained for large numbers of random trees show that one of the heuristics we have designed leads to good results in practice.

A possible future direction is to consider so-called *non-linear strategies* [8]. Although in this work we have considered solutions expressed as leaf orderings (called *linear strategies* in [8]), 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 may be exponential in the number of tree leaves. In [8], it is shown that for *read-once* queries 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 for *shared* queries (see G), thus motivating the investigation of non-linear strategies for such queries. Another direction is to study the problem for *periodic* query evaluations. In this work we have considered a single query evaluation, but in practice queries on sensor data streams are evaluated periodically. As a result, data items acquired from previous query evaluations may be re-used for the current evaluation, depending on predicate time-windows and the query evaluation period. The problem is to determine a schedule that minimizes the expected cost of the query evaluation in the long run. The computation of the cost of a given schedule is more complex due to the need to account for data items "left over" from previous query evaluations, and we expect the problem to be more computationally challenging than that studied in this work.

References

References

[1] E. Miluzzo, N. D. Lane, K. Fodor, R. Peterson, H. Lu, M. Musolesi, S. B. Eisenman, X. Zheng, A. T. Campbell, Sensing meets mobile social networks: The design, implementation and evaluation of the cenceme application, in: Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems, SenSys '08, ACM, 2008, pp. 337–350. doi:10.1145/1460412.1460445.

[2] I. Mohomed, A. Misra, M. Ebling, 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] Y. Jiang, H. Qiu, M. McCartney, W. Halfond, F. Bai, D. Grimm, R. Govindan, Flexible and Efficient Sensor Fusion for Automotive Apps, Technical Report 13-939, Univ. of Southern California, http://www.cs.usc.edu/assets/007/89156.pdf (2013).

[4] S. Gaonkar, J. Li, R. R. Choudhury, L. Cox, A. Schmidt, Micro-blog: Sharing and querying content through mobile phones and social participation, in: Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services, MobiSys '08, ACM, 2008, pp. 174–186. doi:10.1145/1378600.1378620.

[5] L. Lim, A. Misra, T. Mo, Adaptive Data Acquisition Strategies for Energy-Efficient Smartphone-based Continuous Processing of Sensor Streams, Distributed Parallel Databases 31 (2) (2013) 321–351.

[6] The SHIMMER sensor platform, http://shimmer-research.com (2013).

[7] D. E. Smith, Controlling backward inference, Artificial Intelligence 39 (2) (1989) 145—208.

[8] R. Greiner, R. Hayward, M. Jankowska, M. Molloy, Finding Optimal Satisficing Strategies for And-Or Trees, Artificial Intelligence 170 (1) (2006) 19–58.

[9] T. Ünlüyurt, Sequential testing of complex systems: a review, Discrete Applied Mathematics 142 (1-3) (2004) 189–205.

[10] M. Charikar, R. Fagin, V. Guruswami, J. Kleinberg, P. Raghavan, A. Sahai, Query strategies for priced information, J. Comput. Syst. Sci. 64 (4) (2002) 785–819. doi:10.1006/jcss.2002.1828.

[11] F. Cicalese, E. Laber, A. Medeiros Saettler, Decision Trees for the efficient evaluation of discrete functions: worst case and expected case analysis, ArXiv e-printsarXiv:1309.2796.

[12] F. Cicalese, E. S. Laber, On the competitive ratio of evaluating priced functions, J. ACM 58 (3) (2011) 9:1–9:40. doi:10.1145/1970392.1970393.

[13] D. Golovin, A. Krause, D. Ray, Near-optimal bayesian active learning with noisy observations, in: J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, A. Culotta (Eds.), Advances in Neural Information Processing Systems 23, Neural Information Processing Systems Foundation, 2010, pp. 766–774.

[14] G. Bellala, S. Bhavnani, C. Scott, Group-based active query selection for rapid diagnosis in time-critical situations, IEEE Transactions on Information Theory 58 (1) (2012) 459–478. doi:10.1109/TIT.2011.2169296.

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

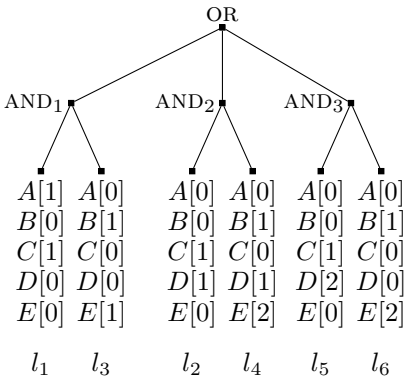


Figure 11: Example *multi-stream* DNF tree.

A Example cost evaluation for a *Multi-stream* DNF tree

Figure 11 shows a *multi-stream* DNF tree with three AND nodes, for five streams A, B, C, D, and E. Each leaf requires one or two data items from multiple streams. Leaves are labeled l_1 to l_6, 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 *multi-stream* DNF trees for *shared* queries. In particular, computing the cost of a schedule is much more complicated than in the *read-once single-stream* case due to inter-leaf dependencies and to the multiple access to several streams. Let $\mathcal{C}_j$ be the cost of evaluating leaf l_j, and $\mathcal{C}$ the overall cost of the schedule. We consider the 6 leaves one by one, in order:

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

Leaf l_2 – This is the first leaf in its AND, no AND has been fully evaluated so far and since l_1 is always evaluated the first data item from stream C required by l_2 is always available for free. l_2 is the first encountered leaf that requires stream D. Therefore, l_2 is always evaluated, requiring a data item from stream D: $\mathcal{C}_2 = c(D)$.

Leaf l_3 – This is the second leaf in its AND, no AND has been fully evaluated so far, and l_3 is the first encountered leaf that requires stream B and stream E. Therefore, a data item from B and a data item from E are acquired if and only if l_1 evaluates to TRUE: $\mathcal{C}_3 = p_1(c(B) + c(E))$.

Leaf l_4 – This is the second leaf in its AND, and it requires a data item from stream D. This data item has already been acquired by l_2 and is available "for free" because l_2 is always evaluated. l_4 also requires a data item from stream B and a data item from stream E. These data items are also required by leaf l_3 in AND_1, which has been fully evaluated. 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 (with probability p_2), l_4 must be evaluated thus requiring the data item from stream B and the first data item from stream E. Finally, l_3 is the first encountered leaf that requires the second data item from stream E, so if l_2 has evaluated to TRUE, then we must also require the second data item from stream E. We obtain $\mathcal{C}_4 = p_2[(1 - p_1)(c(B) + c(E)) + c(E)]$.

Leaf l_5 – Since l_2 is always evaluated, the data item from stream C and the first data item from stream D required by l_5 are always available for free. l_5 is the first encountered leaf that requires the second data item from stream D, so if AND_1 and AND_2 have evaluated to FALSE, then the second data item from D is acquired by l_5. We obtain $\mathcal{C}_5 = (1 - p_1p_3)(1 - p_2p_4)c(D)$

Leaf l_6 – This is the second leaf in its AND, and AND_1 and AND_2 have been fully evaluated so far. However, one of the leaves of AND_1, l_3, and one of the leaves of AND_2, l_4, require a data item from stream B and a data item from stream E that are needed by l_6. Leaf l_6

must be evaluated if l_5 has evaluated to TRUE (with probability p_5) and must acquire a data item from stream B and the first data item from stream E if and only if l_3 and l_4 have not been evaluated (with probability $(1-p_1)(1-p_2)$). l_6 must also acquire the second data item from stream E if and only if l_4 has not been evaluated (with probability $(1-p_2)$). We obtain $\mathcal{C}_6 = p_5[(1-p_1)(1-p_2)(c(B)+c(E)) + (1-p_2)c(E)]$.

Overall, we obtain the cost of the schedule:

$$\begin{aligned}\mathcal{TC} &= c(A) + c(C) + [p_1 + (1-p_1)(p_2 + p_5(1-p_2))]c(B) + \\ &+ (1 + (1-p_1p_3)(1-p_2p_4))c(D) + [p_1 + (2-p_1)(p_2 + p_5(1-p_2))]c(E)\end{aligned}$$

Given the complexity of the above cost computation, one might expect the PAOTR problem to be NP-complete in the *shared multi-stream* case. We confirm this expectation for AND trees in Section 5.3.1.

B Proof of Proposition 2

Proof. We prove the proposition by contradiction. Consider an AND-tree and two leaves in the tree l_1 and l_2 that require data items from the same stream s such that $d_1^s > d_2^s$. We terms these leaves "inverted" because the earlier one, l_1, requires more data items than the later one, l_2. Assume that there is an optimal schedule ξ in which l_1 is scheduled before l_2. Without loss of generality, we assume that l_1 is the first leaf in the schedule that is part of an inverted pair of leaves (if not, consider the earliest such leaf). Evaluating l_2 has always cost zero in this schedule because all data items required by l_2 are also required by l_1.

The sequence of leaves in ξ can be written as: $l_{b_1}, \ldots, l_{b_t}, l_1, l_{m_1}, \ldots, l_{m_u}, l_2, l_{a_1}, \ldots, l_{a_v}$. The cost $\mathcal{C}$ of ξ can be written:

$$\mathcal{C} = X + \mathcal{P}_b \cdot (d_1^s - d_{LB}^s)c(s) + \mathcal{P}_b \cdot p_1 \cdot Y + \mathcal{P}_b \cdot p_1 \cdot \mathcal{P}_m \cdot 0 + \mathcal{P}_b \cdot p_1 \cdot \mathcal{P}_m \cdot p_2 \cdot Z$$

where

- $\mathcal{P}_b = \prod_{i=1}^{t} p_{b_i}$ and $\mathcal{P}_m = \prod_{i=1}^{u} p_{m_i}$;
- X is the expected cost of evaluating leaves $l_{b_1}, \ldots, l_{b_t}$ in that order;
- Y is the expected cost of evaluating leaves $l_{m_1}, \ldots, l_{m_u}$ in that order if leaves $l_{b_1}, \ldots, l_{l_t}$ and l_1 all evaluate to TRUE;
- Z is the expected cost of evaluating leaves $l_{a_1}, \ldots, l_{a_v}$ in that order if leaves $l_{b_1}, \ldots, l_{b_t}$, l_1, $l_{l_1}, \ldots, l_{m_u}$, and l_2 all evaluated to TRUE;
- $d_{LB}^s = \max_{i=1,\ldots,t}(d_{b_i}^s)$, or the number of elements of stream s that have been acquired after evaluating leaves $l_{b_1}, \ldots, l_{b_t}$.

Because l_1 and l_2 are the first two inverted leaves in ξ, $d_1^s - d_{LB}^s$ is non-negative (otherwise a leaf among $l_{b_1}, \ldots, l_{b_t}$ and leaf l_1 would be inverted).

We now construct another schedule, ξ', as $l_{b_1}, \ldots, l_{b_t}, l_2, l_1, l_{m_1}, \ldots, l_{m_u}, l_{a_1}, \ldots, l_{a_v}$. The expected cost $\mathcal{C}$' of ξ' can then be written as:

$$\mathcal{C}' = X + \mathcal{P}_b \cdot (d_2^s - d_{LB}^s)c(s) + \mathcal{P}_b \cdot p_2(d_1^s - d_2^s)c(s) + \mathcal{P}_b \cdot p_2 \cdot p_1 \cdot Y + \mathcal{P}_b \cdot p_2 \cdot p_1 \cdot \mathcal{P}_m \cdot Z$$

Because l_1 and l_2 are the first two inverted leaves in ξ, $d_2^s - d_{LB}^s$ is non-negative (otherwise a leaf among $l_{b_1}, \ldots, l_{b_t}$ and leaf l_2 would be inverted). Computing the difference of the costs of both schedules yields:

$$\mathcal{C} - \mathcal{C}' = \mathcal{P}_b(1-p_2)\left((d_1^s - d_2^s)c(s) + p_1 Y\right)$$

$\mathcal{C} - \mathcal{C}'$ is strictly positive because all costs are positives, all probabilities are between 0 and 1, and because $d_1^s > d_2^s$ by assumption. This contradicts the optimality of ξ. □

C Proof of Theorem 1

Proof. 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, let us pick a schedule, ξ_{opt}, which has the longest prefix $\mathbb{P}$ in common with schedule ξ_{greedy}. We consider the first decision (i.e., one recursive call to the algorithm) taken by Algorithm 1 that schedules a leaf that does not belong to $\mathbb{P}$. Let k be the number of leaves scheduled by this decision, and let us denote them $l_{\sigma(1)}$, ..., $l_{\sigma(k)}$, scheduled in this order. Recall each call to the SingleStreamGreedy algorithm schedules a sequence of leaves that all require data items from the same stream. Furthermore, the scheduled sequence of leaves is a sub-sequence of the ordered sequence of all leaves that require data items from that stream, sorted by increasing number of data items required. Without loss of generality, we assume that $l_{\sigma(1)}$, ..., $l_{\sigma(k)}$ all require items from stream 1. The first of these leaves may belong to $\mathbb{P}$ (as the last leaf occurrences in $\mathbb{P}$). Let $\mathbb{P}'$ be equal to $\mathbb{P}$ minus the leaves $l_{\sigma(1)}$, ..., $l_{\sigma(k)}$. Then, ξ_{greedy} can be written as:

$$\xi_{greedy} = \mathbb{P}', l_{\sigma(1)}, ..., l_{\sigma(k)}, \mathbb{S}. \tag{11}$$

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 $L_1 l_{\sigma(1)} L_2 l_{\sigma(2)} ... L_k l_{\sigma(k)}$, where each sequence of leaves L_i, $1 \leq i \leq k$, can be empty. Note that, because of Theorem 2, and because the sequence $l_{\sigma(1)}$, ..., $l_{\sigma(k)}$ is a sub-sequence of the the list of all leaves requiring data items from that stream sorted by increasing number of data items required, none of the L_i sequences can contain a leaf requiring elements from stream 1. Therefore,

$$\xi_{opt} = \mathbb{P}', \mathbb{Q}, \mathbb{R} \text{ where } \mathbb{Q} = L_1 l_{\sigma(1)} L_2 l_{\sigma(2)} ... L_k l_{\sigma(k)} \tag{12}$$

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

$$\xi_{new} = \mathbb{P}', NewOrder, \mathbb{R} \text{ where } NewOrder = l_{\sigma(1)}, ..., l_{\sigma(k)}, L_1, ..., L_k \tag{13}$$

$\mathbb{P}'$, $l_{\sigma(1)}$, ..., $l_{\sigma(k)}$ is a prefix to both ξ_{greedy} and ξ_{new}. This prefix is strictly larger than $\mathbb{P}$ (since $\mathbb{P}$ does not contain $l_{\sigma(k)}$). Therefore, if the cost of ξ_{new} is not greater than that of ξ_{opt}, ξ_{new} is optimal and has a longer prefix in common with ξ_{greedy} than ξ_{new}, which would contradict the definition of ξ_{opt}. We obtain this contradiction by computing the cost of ξ_{new} and showing that it is no larger than that of ξ_{opt}.

Cost notations – To ease the writing of the proof we introduce several notations. If $\mathbb{X}$ is a partial leaf schedule, $P(\mathbb{X})$ denotes the probability that all leaves in $\mathbb{X}$ evaluates to TRUE. In other words, $P(\mathbb{X}) = \prod_{l_i \in \mathbb{X}} p_i$. Let $\mathbb{X}$ and $\mathbb{Y}$ be two disjoint (partial) leaf schedules, i.e., they do not have any leaf in common, such that $\mathbb{X}$ is evaluated right before $\mathbb{Y}$. Then $Cost(\mathbb{Y} \mid \mathbb{X})$ denotes the cost of evaluating $\mathbb{Y}$, assuming that all leaves in $\mathbb{X}$ have evaluated to TRUE. Of course, $Cost(\mathbb{Y} \mid \mathbb{X})$ takes into account all data items acquired during the successful evaluation of $\mathbb{X}$. With these notations, we can now give the costs of ξ_{new} and ξ_{opt} based on their definitions as sequences of partial leaf schedules in Equations (12) and (13):

$$\begin{aligned} Cost(\xi_{opt}) &= Cost(\mathbb{P}') + P(\mathbb{P}')\,Cost(\mathbb{Q} \mid \mathbb{P}') \\ &\quad + P(\mathbb{P}')P(\mathbb{Q})\,Cost(\mathbb{R} \mid \mathbb{P}', \mathbb{Q}) \\ Cost(\xi_{new}) &= Cost(\mathbb{P}') + P(\mathbb{P}')\,Cost(NewOrder \mid \mathbb{P}') \\ &\quad + P(\mathbb{P}')P(NewOrder)\,Cost(\mathbb{R} \mid \mathbb{P}', NewOrder) \end{aligned}$$

Because $\mathbb{Q}$ and $NewOrder$ contain exactly the same leaves, $P(\mathbb{Q}) = P(NewOrder)$ and $Cost(\mathbb{R} \mid \mathbb{P}', \mathbb{Q}) = Cost(\mathbb{R} \mid \mathbb{P}', NewOrder)$. Therefore,

$$Cost(\xi_{opt}) - Cost(\xi_{new}) = P(\mathbb{P}')\,(Cost(\mathbb{Q} \mid \mathbb{P}') - Cost(NewOrder \mid \mathbb{P}')) \tag{14}$$

From what precedes, it now suffices to show that $Cost(\mathbb{Q} \mid \mathbb{P}') - Cost(NewOrder \mid \mathbb{P}') \geq 0$ to prove the theorem.

Initial mathematical formulation – We use Proposition 2 to define notations that make it possible to obtain a simple expression for the quantity in Equation 14. Consider a stream S and two leaves l_i and l_j that require, respectively, $d_i^S > 0$ and $d_j^S > 0$ items from stream S, with $d_i^S < d_j^S$. Then, according to Proposition 2, l_i is always evaluated before l_j in an optimal schedule. The SingleStreamGreedy algorithm also schedules l_i before l_j. If there does not exist any leaf l_k requiring $d_k^S \in [d_i^S; d_j^S]$ elements from stream S, then each time l_j is evaluated, exactly $d_j^S - d_i^S$ items are acquired from stream S, because the last d_i^S elements of stream S were acquired when l_i was evaluated. In this case, we define a_i as the number of data items that must be acquired when evaluating leaf l_i. Formally,

$$a_i = d_i^S - \max\left\{d_j^S \mid d_j^S < d_i^S\right\}$$

Remark: One should note that we can assume without loss of generality that the AND tree does not contain two leaves requiring the exact same number of items from the same stream. If such two leaves exist, then one replaces them by a single leaf with the same data item requirement and with a probability of success that is the product of the probability of success of the two original leaves. This is because once one of the two original leaves has been evaluated then the other one can be evaluated for free.

To ease the writing of the proof, we index the leaves in L_1, ..., L_k according to the stream from which they require data items, and introduce the following additional notations. Let N_i be the number of leaves in $L_1 \cup \ldots \cup L_k$ that require data items from stream i and $l_{i,j}$ be the j-th of these leaves. We then extend the notations defined in Section 3 as follows: the probability of success of $l_{i,j}$ is $p_{i,j}$, $l_{i,j}$ requires $d_{i,j}^{s_i}$ elements from stream s_i, etc. $\mu_{(i,j)}$ is the index of the leaf sequence L_p to which leaf $l_{i,j}$ belongs: $l_{i,j} \in L_{\mu_{(i,j)}}$. $Q_{i,j}$ is the product of the success probabilities of the leaves that precede $l_{i,j}$ in $L_{\mu_{(i,j)}}$, Q_m is the product of the success probabilities of all the leaves in L_m, and $\mathcal{Q}_m = \prod_{n=1}^{m} Q_n$. Finally, we define $P_m = \prod_{n=1}^{m} p_{\sigma(n)}$. With these notations we can now write $Cost(NewOrder \mid \mathbb{P}')$ as:

$$\begin{aligned} Cost(NewOrder \mid \mathbb{P}') &= \sum_{m=1}^{k} \left(\prod_{n=1}^{m-1} p_{\sigma(n)}\right) a_{\sigma(m)} \\ &\quad + \sum_{i=2}^{S}\sum_{j=1}^{N_i} \left(\prod_{m=1}^{k} p_{\sigma(m)}\right)\left(\prod_{m=1}^{\mu_{(i,j)}-1} Q_m\right) Q_{i,j} a_{i,j} c(s_i) \\ &= \sum_{m=1}^{k} P_{m-1} a_{\sigma(m)} + \sum_{i=2}^{S}\sum_{j=1}^{N_i} P_k \mathcal{Q}_{\mu_{(i,j)}-1} Q_{i,j} a_{i,j} c(s_i) \ , \end{aligned}$$

and $Cost(\mathbb{Q} \mid \mathbb{P}')$ as:

$$\begin{aligned} Cost(\mathbb{Q} \mid \mathbb{P}') &= \sum_{m=1}^{k} \left(\prod_{n=1}^{m-1} p_{\sigma(n)}\right)\left(\prod_{n=1}^{m} Q_n\right) a_{\sigma(m)} \\ &\quad + \sum_{i=2}^{S}\sum_{j=1}^{N_i} \left(\prod_{m=1}^{\mu_{(i,j)}-1} p_{\sigma(m)}\right)\left(\prod_{m=1}^{\mu_{(i,j)}-1} Q_m\right) Q_{i,j} a_{i,j} c(s_i) \\ &= \sum_{m=1}^{k} P_{m-1} \mathcal{Q}_m a_{\sigma(m)} + \sum_{i=2}^{S}\sum_{j=1}^{N_i} P_{\mu_{(i,j)}-1} \mathcal{Q}_{\mu_{(i,j)}-1} Q_{i,j} a_{i,j} c(s_i) \ . \end{aligned}$$

Therefore:

$$\begin{aligned} \mathit{Cost}(\mathbb{Q} \mid \mathbb{P}') - \mathit{Cost}(\mathit{NewOrder} \mid \mathbb{P}') &= \sum_{m=1}^{k} P_{m-1}\mathcal{Q}_m a_{\sigma(m)} \\ &+ \sum_{i=2}^{S}\sum_{j=1}^{N_i} P_{\mu_{(i,j)}-1}\mathcal{Q}_{\mu_{(i,j)}-1}Q_{i,j}a_{i,j}c(s_i) \\ &- \left(\sum_{m=1}^{k} P_{m-1}a_{\sigma(m)} + \sum_{i=2}^{S}\sum_{j=1}^{N_i} P_k\mathcal{Q}_{\mu_{(i,j)}-1}Q_{i,j}a_{i,j}c(s_i)\right) \\ &= \sum_{m=1}^{k} P_{m-1}(\mathcal{Q}_m - 1)a_{\sigma(m)} \\ &+ \sum_{i=2}^{S}\sum_{j=1}^{N_i} \left(P_{\mu_{(i,j)}-1} - P_k\right)\mathcal{Q}_{\mu_{(i,j)}-1}Q_{i,j}a_{i,j}c(s_i)\ . \end{aligned}$$

We introduce two additional notations:

$$\alpha_{(i,j)} = \mathcal{Q}_{\mu_{(i,j)}-1}\left(P_{\mu_{(i,j)}-1} - P_k\right)Q_{i,j}\ \text{, and}$$

$$A = \frac{\sum_{m=1}^{k} P_{m-1}a_{\sigma(m)}}{1-P_k}\ ,$$

so that we can finally write the expression for the difference of the two costs:

$$\mathit{Cost}(\mathbb{Q} \mid \mathbb{P}') - \mathit{Cost}(\mathit{NewOrder} \mid \mathbb{P}') = \left(\sum_{m=1}^{k} P_{m-1}\left(\mathcal{Q}_m - 1\right)a_{\sigma(m)}\right) + \left(\sum_{i=2}^{S}\sum_{j=1}^{N_i}\alpha_{(i,j)}a_{i,j}c(s_i)\right)\ . \tag{15}$$

Accounting for the algorithm's scheduling decisions – The best decision for the SingleStreamGreedy algorithm was to evaluate at once the leaf sequence $l_{\sigma(1)}, ..., l_{\sigma(k)}$. Therefore, as far as the algorithm is concerned, this was a better decision than evaluating any sequence of leaves from any other stream. More formally, for any stream i, $2 \leq i \leq S$, and the set of the first j leaves of that stream, $1 \leq j \leq N_i$, we have:

$$\frac{\left(\sum_{m=1}^{k}\left(\prod_{n=1}^{m-1} p_{\sigma(n)}\right)a_{\sigma(m)}\right)}{\left(1-\prod_{m=1}^{k} p_{\sigma(m)}\right)}\left(1-\prod_{l=1}^{j} p_{i,l}\right) \leq \left(\sum_{l=1}^{j}\left(\prod_{r=1}^{l-1} p_{i,r}\right)a_{i,l}c(s_i)\right)\ .$$

These equations express the fact that these other sequence of leaves of a *Ratio* value (see Algorithm 1) lower than that of the sequence scheduled by the algorithm, and can be rewritten as:

$$\textsc{Ineq}(i,j): \qquad A\left(1-\prod_{l=1}^{j} p_{i,l}\right) \leq \left(\sum_{l=1}^{j}\left(\prod_{r=1}^{l-1} p_{i,r}\right)a_{i,l}c(s_i)\right)\ . \tag{16}$$

Determining multiplying coefficients – To prove the theorem we combine the Inequalities (16) obtained for different values of i and j. The idea is to follow a variable elimination process. $\textsc{Ineq}(i, N_i)$ is the only inequality in which a_{i,N_i} appears. We multiply $\textsc{Ineq}(i, N_i)$ by a value λ_{i,N_i} such that, in the resulting inequality, the coefficient of a_{i,N_i} is the same than in

Equation (15). Next, we multiply $\textsc{Ineq}(i, N_i - 1)$ by a value λ_{i,N_i-1} such that when adding the resulting inequality to the one previously obtained, the coefficient of a_{i,N_i-1} is the same than in Equation (15), and so on. This process can be done independently for the different streams as $\textsc{Ineq}(i, j)$ only contains terms relative to stream i.

We define the $\lambda_{i,j}$'s are defined as follows.

$$\lambda_{i,j} = \begin{cases} \dfrac{\alpha_{(i,N_i)}}{\prod_{l=1}^{N_i-1} p_{i,l}} & \text{if } j = N_i \ , \\ \dfrac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} - \dfrac{\alpha_{(i,j+1)}}{\prod_{l=1}^{j} p_{i,l}} & \text{otherwise.} \end{cases}$$

We will later show that this choice of multipliers enable us to achieve our goal. However, as we want to use the $\lambda_{i,j}$'s as multiplying coefficients for inequalities, we must first show that they are all non-negative. This is evident for the λ_{i,N_i}'s. Let us consider $\lambda_{i,j}$ for $j \in [1; N_i - 1]$:

$$\begin{aligned} \lambda_{i,j} &= \frac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} - \frac{\alpha_{(i,j+1)}}{\prod_{l=1}^{j} p_{i,l}} \\ &= \frac{1}{\prod_{l=1}^{j-1} p_{i,l}} \left(\prod_{m=1}^{\mu_{(i,j)}-1} Q_m \right) \left(\prod_{m=1}^{\mu_{(i,j)}-1} p_{\sigma(m)} \right) \left(1 - \prod_{m=\mu_{(i,j)}}^{k} p_{\sigma(m)} \right) Q_{i,j} \\ &\quad - \frac{1}{\prod_{l=1}^{j} p_{i,l}} \left(\prod_{m=1}^{\mu_{(i,j+1)}-1} Q_m \right) \left(\prod_{m=1}^{\mu_{(i,j+1)}-1} p_{\sigma(m)} \right) \left(1 - \prod_{m=\mu_{(i,j+1)}}^{k} p_{\sigma(m)} \right) Q_{i,j+1} \\ &= \frac{1}{\prod_{l=1}^{j-1} p_{i,l}} \left(\prod_{m=1}^{\mu_{(i,j)}-1} Q_m p_{\sigma(m)} \right) \\ &\quad \times \left[\left(1 - \prod_{m=\mu_{(i,j)}}^{k} p_{\sigma(m)} \right) Q_{i,j} - \left(\prod_{m=\mu_{(i,j)}}^{\mu_{(i,j+1)}-1} Q_m p_{\sigma(m)} \right) \left(1 - \prod_{m=\mu_{(i,j+1)}}^{k} p_{\sigma(m)} \right) \frac{Q_{i,j+1}}{p_{i,j}} \right] \end{aligned}$$

Let us first consider the case $\mu_{(i,j+1)} = \mu_{(i,j)}$. Then the above equation can be rewritten:

$$\lambda_{i,j} = \frac{1}{\prod_{l=1}^{j-1} p_{i,l}} \left(\prod_{m=1}^{\mu_{(i,j)}-1} Q_m p_{\sigma(m)} \right) \left(1 - \prod_{m=\mu_{(i,j)}}^{k} p_{\sigma(m)} \right) \left[Q_{i,j} - \frac{Q_{i,j+1}}{p_{i,j}} \right]$$

By definition, $Q_{i,j+1}$ is the product of the probabilities of success of all the leaves that are evaluated before the leaf $l_{i,j+1}$ is evaluated. By definition of the numbering of the leaves, this includes at least all the leaves that are evaluated before leaf $l_{i,j}$ is evaluated and leaf $l_{i,j}$. As all probabilities are less than or equal to 1, this implies that $Q_{i,j+1} \leq Q_{i,j} p_{i,j}$, and therefore that $\lambda_{i,j} \geq 0$.

We now consider the other case: $\mu_{(i,j+1)} > \mu_{(i,j)}$. Then, $Q_{\mu_{(i,j)}}$ is of the form $Q_{i,j} p_{i,j} X$ where X is the product of the probabilities of success of the leaves appearing in $L_{\mu_{(i,j)}}$ after the leaf $l_{i,j}$. Therefore, $Q_{i,j} p_{i,j} \geq Q_{\mu_{(i,j)}}$. As for all $i \in [1; S]$, $0 \leq p_{\sigma(i)} \leq 1$, $\prod_{m=\mu_{(i,j+1)}}^{k} p_{\sigma(m)} \geq \prod_{m=\mu_{(i,j)}}^{k} p_{\sigma(m)}$ and

$1 - \prod_{m=\mu_{(i,j+1)}}^{k} p_{\sigma(m)} \leq 1 - \prod_{m=\mu_{(i,j)}}^{k} p_{\sigma(m)}$, $p_{\sigma(\mu_{(i,j)})}(\prod_{m=\mu_{(i,j)}+1}^{\mu_{(i,j+1)}-1} Q_m p_{\sigma(m)})Q_{i,j+1} \leq 1$ because it is a product of probabilities. Using these inequalities, we have:

$$\begin{aligned}
&\left(1 - \prod_{m=\mu_{(i,j)}}^{k} p_{\sigma(m)}\right) Q_{i,j} - \left(\prod_{m=\mu_{(i,j)}}^{\mu_{(i,j+1)}-1} Q_m p_{\sigma(m)}\right) \left(1 - \prod_{m=\mu_{(i,j+1)}}^{k} p_{\sigma(m)}\right) \frac{Q_{i,j+1}}{p_{i,j}} &\geq \\
&\left(1 - \prod_{m=\mu_{(i,j)}}^{k} p_{\sigma(m)}\right) Q_{i,j} - \left(\prod_{m=\mu_{(i,j)}}^{\mu_{(i,j+1)}-1} Q_m p_{\sigma(m)}\right) \left(1 - \prod_{m=\mu_{(i,j)}}^{k} p_{\sigma(m)}\right) \frac{Q_{i,j+1}}{p_{i,j}} &= \\
&\left(1 - \prod_{m=\mu_{(i,j)}}^{k} p_{\sigma(m)}\right) \left(Q_{i,j} - \left(\prod_{m=\mu_{(i,j)}}^{\mu_{(i,j+1)}-1} Q_m p_{\sigma(m)}\right) \frac{Q_{i,j+1}}{p_{i,j}}\right) &\geq \\
&\left(1 - \prod_{m=\mu_{(i,j)}}^{k} p_{\sigma(m)}\right) \left(Q_{i,j} - Q_{\mu_{(i,j)}} \left(p_{\sigma(\mu_{(i,j)})}(\prod_{m=\mu_{(i,j)}+1}^{\mu_{(i,j+1)}-1} Q_m p_{\sigma(m)})Q_{i,j+1}\right) \frac{1}{p_{i,j}}\right) &\geq \\
&\left(1 - \prod_{m=\mu_{(i,j)}}^{k} p_{\sigma(m)}\right) \left(Q_{i,j} - Q_{\mu_{(i,j)}} \frac{1}{p_{i,j}}\right) &\geq 0
\end{aligned}$$

Therefore, all the $\lambda_{i,j}$'s are non-negative.

Combining the inequalities – For a given couple of values (i, j), with $2 \leq i \leq S$ and $1 \leq j \leq N_i$, let $\textsc{Ineq}(i, j)$ be Inequality (16) defined for (i, j). Because all the $\lambda_{i,j}$'s are non-negative, we can form the inequality:

$$\sum_{i=2}^{S} \sum_{j=1}^{N_i} (\lambda_{i,j} \times \textsc{Ineq}(i, j)) \tag{17}$$

We now show that Inequality (17) leads to:

$$Cost(\mathbb{Q} \mid \mathbb{P}') - Cost(NewOrder \mid \mathbb{P}') \geq \sum_{m=1}^{k} P_{m-1} \left(\mathcal{Q}_m - 1\right) a_{\sigma(m)} + A \sum_{i=2}^{S} \left(\sum_{j=1}^{N_i} \alpha_{(i,j)} \left(1 - p_{i,j}\right) \right) \tag{18}$$

To prove the Inequality (18), we consider the terms relative to stream i in Inequality (17):

$$\begin{aligned}
&\sum_{j=1}^{N_i} (\lambda_{i,j} \times \textsc{Ineq}(i, j)) \qquad \Leftrightarrow \\
A \sum_{j=1}^{N_i} \lambda_{i,j} \left(1 - \prod_{l=1}^{j} p_{i,l}\right) \quad &\leq \quad \sum_{j=1}^{N_i} \lambda_{i,j} \left(\sum_{l=1}^{j} \left(\prod_{r=1}^{l-1} p_{i,r} \right) a_{i,l} c(s_i) \right)
\end{aligned} \tag{19}$$

We start by considering the left-hand side of this inequality.

$$
\begin{aligned}
&\sum_{j=1}^{N_i} \lambda_{i,j} \left(1 - \prod_{l=1}^{j} p_{i,l}\right) && = \\
&\left(\sum_{j=1}^{N_i-1} \lambda_{i,j} \left(1 - \prod_{l=1}^{j} p_{i,l}\right)\right) + \lambda_{i,N_i} \left(1 - \prod_{l=1}^{N_i} p_{i,l}\right) && = \\
&\left(\sum_{j=1}^{N_i-1} \left(\frac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} - \frac{\alpha_{(i,j+1)}}{\prod_{l=1}^{j} p_{i,l}}\right) \left(1 - \prod_{l=1}^{j} p_{i,l}\right)\right) + \frac{\alpha_{(i,N_i)}}{\prod_{l=1}^{N_i-1} p_{i,l}} \left(1 - \prod_{l=1}^{N_i} p_{i,l}\right) && = \\
&\left(\sum_{j=1}^{N_i-1} \frac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} \left(1 - \prod_{l=1}^{j} p_{i,l}\right)\right) - \left(\sum_{j=1}^{N_i-1} \frac{\alpha_{(i,j+1)}}{\prod_{l=1}^{j} p_{i,l}} \left(1 - \prod_{l=1}^{j} p_{i,l}\right)\right) + \frac{\alpha_{(i,N_i)}}{\prod_{l=1}^{N_i-1} p_{i,l}} \left(1 - \prod_{l=1}^{N_i} p_{i,l}\right) && = \\
&\left(\sum_{j=1}^{N_i-1} \frac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} \left(1 - \prod_{l=1}^{j} p_{i,l}\right)\right) - \left(\sum_{j=2}^{N_i} \frac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} \left(1 - \prod_{l=1}^{j-1} p_{i,l}\right)\right) + \frac{\alpha_{(i,N_i)}}{\prod_{l=1}^{N_i-1} p_{i,l}} \left(1 - \prod_{l=1}^{N_i} p_{i,l}\right) && = \\
&\left(\sum_{j=1}^{N_i} \frac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} \left(1 - \prod_{l=1}^{j} p_{i,l}\right)\right) - \left(\sum_{j=2}^{N_i} \frac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} \left(1 - \prod_{l=1}^{j-1} p_{i,l}\right)\right) && = \\
&\alpha_{(i,1)} \left(1 - p_{i,1}\right) + \left(\sum_{j=2}^{N_i} \frac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} \left(1 - \prod_{l=1}^{j} p_{i,l}\right)\right) - \left(\sum_{j=2}^{N_i} \frac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} \left(1 - \prod_{l=1}^{j-1} p_{i,l}\right)\right) && = \\
&\alpha_{(i,1)} \left(1 - p_{i,1}\right) + \sum_{j=2}^{N_i} \frac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} \left(\prod_{l=1}^{j-1} p_{i,l} - \prod_{l=1}^{j} p_{i,l}\right) && = \\
&\alpha_{(i,1)} \left(1 - p_{i,1}\right) + \sum_{j=2}^{N_i} \alpha_{(i,j)} \left(1 - p_{i,j}\right) && = \\
&\sum_{j=1}^{N_i} \alpha_{(i,j)} \left(1 - p_{i,j}\right) \ . &&
\end{aligned}
$$

Therefore,

$$
A \sum_{j=1}^{N_i} \lambda_{i,j} \left(1 - \prod_{l=1}^{j} p_{i,l}\right) = A \left(\sum_{j=1}^{N_i} \alpha_{(i,j)} \left(1 - p_{i,j}\right)\right) \ . \tag{20}
$$

We now focus on the right-hand side of the inequality:

$$
\begin{aligned}
&\sum_{j=1}^{N_i} \lambda_{i,j} \left(\sum_{l=1}^{j} \left(\prod_{r=1}^{l-1} p_{i,r} \right) a_{i,l} c(s_i) \right) = \\
&\left(\sum_{j=1}^{N_i-1} \lambda_{i,j} \left(\sum_{l=1}^{j} \left(\prod_{r=1}^{l-1} p_{i,r} \right) a_{i,l} c(s_i) \right) \right) + \lambda_{i,N_i} \left(\sum_{l=1}^{N_i} \left(\prod_{r=1}^{l-1} p_{i,r} \right) a_{i,l} c(s_i) \right) = \\
&\left(\sum_{j=1}^{N_i-1} \left(\frac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} - \frac{\alpha_{(i,j+1)}}{\prod_{l=1}^{j} p_{i,l}} \right) \left(\sum_{l=1}^{j} \left(\prod_{r=1}^{l-1} p_{i,r} \right) a_{i,l} c(s_i) \right) \right) \\
&\quad + \frac{\alpha_{(i,N_i)}}{\prod_{l=1}^{N_i-1} p_{i,l}} \left(\sum_{l=1}^{N_i} \left(\prod_{r=1}^{l-1} p_{i,r} \right) a_{i,l} c(s_i) \right) = \\
&\left(\sum_{j=1}^{N_i-1} \frac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} \left(\sum_{l=1}^{j} \left(\prod_{r=1}^{l-1} p_{i,r} \right) a_{i,l} c(s_i) \right) \right) - \left(\sum_{j=1}^{N_i-1} \frac{\alpha_{(i,j+1)}}{\prod_{l=1}^{j} p_{i,l}} \left(\sum_{l=1}^{j} \left(\prod_{r=1}^{l-1} p_{i,r} \right) a_{i,l} c(s_i) \right) \right) \\
&\quad + \frac{\alpha_{(i,N_i)}}{\prod_{l=1}^{N_i-1} p_{i,l}} \left(\sum_{l=1}^{N_i} \left(\prod_{r=1}^{l-1} p_{i,r} \right) a_{i,l} c(s_i) \right) = \\
&\left(\sum_{j=1}^{N_i} \frac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} \left(\sum_{l=1}^{j} \left(\prod_{r=1}^{l-1} p_{i,r} \right) a_{i,l} c(s_i) \right) \right) - \left(\sum_{j=1}^{N_i-1} \frac{\alpha_{(i,j+1)}}{\prod_{l=1}^{j} p_{i,l}} \left(\sum_{l=1}^{j} \left(\prod_{r=1}^{l-1} p_{i,r} \right) a_{i,l} c(s_i) \right) \right) = \\
&\left(\sum_{j=1}^{N_i} \frac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} \left(\sum_{l=1}^{j} \left(\prod_{r=1}^{l-1} p_{i,r} \right) a_{i,l} c(s_i) \right) \right) - \left(\sum_{j=2}^{N_i} \frac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} \left(\sum_{l=1}^{j-1} \left(\prod_{r=1}^{l-1} p_{i,r} \right) a_{i,l} c(s_i) \right) \right) = \\
&\alpha_{(i,1)} a_{i,1} c(s_i) + \left(\sum_{j=2}^{N_i} \frac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} \left(\sum_{l=1}^{j} \left(\prod_{r=1}^{l-1} p_{i,r} \right) a_{i,l} c(s_i) \right) \right) \\
&\quad - \left(\sum_{j=2}^{N_i} \frac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} \left(\sum_{l=1}^{j-1} \left(\prod_{r=1}^{l-1} p_{i,r} \right) a_{i,l} c(s_i) \right) \right) = \\
&\alpha_{(i,1)} a_{i,1} c(s_i) + \sum_{j=2}^{N_i} \frac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} \left(\prod_{r=1}^{j-1} p_{i,r} \right) a_{i,j} c(s_i) = \\
&\sum_{j=1}^{N_i} \alpha_{(i,j)} a_{i,j} c(s_i) \; .
\end{aligned}
$$

Therefore

$$\sum_{j=1}^{N_i} \lambda_{i,j} \left(\sum_{l=1}^{j} \left(\prod_{r=1}^{l-1} p_{i,r} \right) a_{i,l} c(s_i) \right) = \left(\sum_{j=1}^{N_i} \alpha_{(i,j)} a_{i,j} c(s_i) \right) \; . \tag{21}$$

By combining Inequality (19) with Equations (20) and (21), and by summing over all streams, we obtain:

$$A \sum_{i=2}^{S} \left(\sum_{j=1}^{N_i} \alpha_{(i,j)} \left(1 - p_{i,j}\right) \right) \leq \sum_{i=2}^{S} \left(\sum_{j=1}^{N_i} \alpha_{(i,j)} a_{i,j} c(s_i) \right) \; . \tag{22}$$

Using Equation (15) and Inequality (22), we obtain:

$$Cost(\mathbb{Q} \mid \mathbb{P}') - Cost(NewOrder \mid \mathbb{P}') \geq \sum_{m=1}^{k} P_{m-1} \left(\mathcal{Q}_m - 1\right) a_{\sigma(m)} + A \sum_{i=2}^{S} \left(\sum_{j=1}^{N_i} \alpha_{(i,j)} \left(1 - p_{i,j}\right) \right) . \tag{23}$$

Completing the proof – We want to prove that the right-hand side of Inequality (23) is non-negative, i.e., that the following inequality holds:

$$\sum_{m=1}^{k} P_{m-1} \left(\mathcal{Q}_m - 1\right) a_{\sigma(m)} + A \sum_{i=2}^{S} \left(\sum_{j=1}^{N_i} \alpha_{(i,j)} \left(1 - p_{i,j}\right) \right) \geq 0 . \tag{24}$$

Because of Inequality (23), this will enable us to conclude. Let

$$A_n = \sum_{m=1}^{n} P_{m-1} a_{\sigma(m)} .$$

Therefore, $A = \frac{A_k}{(1-P_k)}$. We start by focusing on the first term of Inequality (24). We prove that:

$$\sum_{m=1}^{k} P_{m-1} \left(\mathcal{Q}_m - 1\right) a_{\sigma(m)} \geq A \left(\left(\sum_{i=1}^{k} (\mathcal{Q}_i - \mathcal{Q}_{i-1}) P_{i-1} \right) + P_k (1 - \mathcal{Q}_k) \right) . \tag{25}$$

$$
\begin{aligned}
&\sum_{m=1}^{k} P_{m-1}\,(\mathcal{Q}_m - 1)\,a_{\sigma(m)} \\
&= \sum_{m=1}^{k} (\mathcal{Q}_m - 1)\left(\sum_{n=1}^{m} P_{n-1}a_n - \sum_{n=1}^{m-1} P_{n-1}a_n\right) \\
&= \sum_{m=1}^{k} (\mathcal{Q}_m - 1)(A_m - A_{m-1}) \\
&= \left(\sum_{m=1}^{k} \mathcal{Q}_m(A_m - A_{m-1})\right) - \left(\sum_{m=1}^{k} (A_m - A_{m-1})\right) \\
&= \left(\sum_{m=1}^{k} (\mathcal{Q}_m A_m - \mathcal{Q}_m A_{m-1})\right) - (A_k - A_0) \\
&= \left(\sum_{m=1}^{k} (\mathcal{Q}_m A_m - \mathcal{Q}_{m-1}A_{m-1} + \mathcal{Q}_{m-1}A_{m-1} - \mathcal{Q}_m A_{m-1})\right) - A_k \\
&= \left(\sum_{m=1}^{k} (\mathcal{Q}_m A_m - \mathcal{Q}_{m-1}A_{m-1})\right) + \left(\sum_{m=1}^{k} (\mathcal{Q}_{m-1}A_{m-1} - \mathcal{Q}_m A_{m-1})\right) - A_k \\
&= (\mathcal{Q}_k A_k - \mathcal{Q}_0 A_0) + \left(\sum_{m=1}^{k} (\mathcal{Q}_{m-1} - \mathcal{Q}_m)A_{m-1}\right) - A_k \\
&= \left(\sum_{m=1}^{k} (\mathcal{Q}_{m-1} - \mathcal{Q}_m)A_{m-1}\right) + (\mathcal{Q}_k - 1)A_k = \left(\sum_{m=1}^{k} (\mathcal{Q}_{m-1} - \mathcal{Q}_m)A_{m-1}\right) + (\mathcal{Q}_k - 1)A(1 - P_k)\ .
\end{aligned}
$$

By hypothesis, the best decision for the greedy algorithm was to read at once the leaves a_1, ..., a_k. Therefore, this was better than reading any other sequence of leaves from stream 1. So, for any value of $m \leq k$,

$$A\,(1 - P_{m-1}) \leq A_{m-1}\ .$$

Because $\mathcal{Q}_m = \mathcal{Q}_{m-1}Q_m$ with $Q_m \in [0;1]$, then $\mathcal{Q}_{m-1} - \mathcal{Q}_m \geq 0$. Therefore, we have:

$$\begin{aligned}
&\sum_{m=1}^{k}(\mathcal{Q}_{m-1} - \mathcal{Q}_m)A_{m-1} + A(1-P_k)(\mathcal{Q}_k - 1) \\
&\geq A\left[\left(\sum_{m=1}^{k}(\mathcal{Q}_{m-1} - \mathcal{Q}_m)(1-P_{m-1})\right) + (1-P_k)(\mathcal{Q}_k - 1)\right] \\
&= A\left[\left(\sum_{m=1}^{k}(\mathcal{Q}_{m-1} - \mathcal{Q}_m)\right) - \left(\sum_{m=1}^{k}(\mathcal{Q}_{m-1} - \mathcal{Q}_m)P_{m-1}\right) + (1-P_k)(\mathcal{Q}_k - 1)\right] \\
&= A\left[(\mathcal{Q}_0 - \mathcal{Q}_k) + \left(\sum_{m=1}^{k}(\mathcal{Q}_m - \mathcal{Q}_{m-1})P_{m-1}\right) + (1-P_k)(\mathcal{Q}_k - 1)\right] \\
&= A\left[1 - \mathcal{Q}_k + \left(\sum_{m=1}^{k}(\mathcal{Q}_m - \mathcal{Q}_{m-1})P_{m-1}\right) + (1-P_k)(\mathcal{Q}_k - 1)\right] \\
&= A\left[\left(\sum_{m=1}^{k}(\mathcal{Q}_m - \mathcal{Q}_{m-1})P_{m-1}\right) - P_k(\mathcal{Q}_k - 1)\right] .
\end{aligned}$$

We now focus on the second term of Inequality (24). We prove must prove that:

$$\sum_{i=2}^{S}\sum_{j=1}^{N_i}\alpha_{(i,j)}(1-p_{i,j}) = \left(\sum_{m=1}^{k}\mathcal{Q}_{m-1}P_{m-1}(1-Q_m)\right) - P_k(1-\mathcal{Q}_k) . \tag{26}$$

We have:

$$\begin{aligned}
&\sum_{i=2}^{S}\sum_{j=1}^{N_i}\alpha_{(i,j)}(1-p_{i,j}) \\
&= \sum_{i=2}^{S}\sum_{j=1}^{N_i}\left(\prod_{m=1}^{\mu_{(i,j)}-1}Q_m p_{\sigma(m)}\right)\left(1 - \prod_{m=\mu_{(i,j)}}^{k}p_{\sigma(m)}\right)Q_{i,j}(1-p_{i,j}) \\
&= \sum_{i=2}^{S}\sum_{j=1}^{N_i}\mathcal{Q}_{\mu_{(i,j)}-1}P_{\mu_{(i,j)}-1}\left(1 - \prod_{m=\mu_{(i,j)}}^{k}p_{\sigma(m)}\right)Q_{i,j}(1-p_{i,j}) \\
&= \left(\sum_{i=2}^{S}\sum_{j=1}^{N_i}\mathcal{Q}_{\mu_{(i,j)}-1}P_{\mu_{(i,j)}-1}Q_{i,j}(1-p_{i,j})\right) - \left(\sum_{i=2}^{S}\sum_{j=1}^{N_i}\mathcal{Q}_{\mu_{(i,j)}-1}P_{\mu_{(i,j)}-1}\left(\prod_{m=\mu_{(i,j)}}^{k}p_{\sigma(m)}\right)Q_{i,j}(1-p_{i,j})\right) \\
&= \left(\sum_{i=2}^{S}\sum_{j=1}^{N_i}\mathcal{Q}_{\mu_{(i,j)}-1}P_{\mu_{(i,j)}-1}Q_{i,j}(1-p_{i,j})\right) - \left(\sum_{i=2}^{S}\sum_{j=1}^{N_i}\mathcal{Q}_{\mu_{(i,j)}-1}P_kQ_{i,j}(1-p_{i,j})\right) \\
&= \left(\sum_{i=2}^{S}\sum_{j=1}^{N_i}\mathcal{Q}_{\mu_{(i,j)}-1}P_{\mu_{(i,j)}-1}Q_{i,j}(1-p_{i,j})\right) - \left(P_k\sum_{i=2}^{S}\sum_{j=1}^{N_i}\mathcal{Q}_{\mu_{(i,j)}-1}Q_{i,j}(1-p_{i,j})\right) .
\end{aligned}$$

We concentrate on the second term and its meaning. For any stream i and any of its leaves j, $\mathcal{Q}_{\mu_{(i,j)}-1}Q_{i,j}$ is the probability of success of all the leaves evaluated before the studied leaf (not

considering the leaves of other streams), and $\mathcal{Q}_{\mu_{(i,j)}-1}Q_{i,j}p_{i,j}$ is the same probability right after the evaluation of the studied leaf. Therefore, in the inner sum all terms cancel out except the first one, that is the probability of success if no leaf had been evaluated so far, and the last one, that is the probability of success if all the leaves have been evaluated. Formally, we have:

$$\sum_{i=2}^{S}\sum_{j=1}^{N_i}\mathcal{Q}_{\mu_{(i,j)}-1}Q_{i,j}(1-p_{i,j}) = 1 - \left(\prod_{m=1}^{k} Q_m\right) = 1 - \mathcal{Q}_k \ . \tag{27}$$

Therefore,

$$\begin{aligned}
&\sum_{i=2}^{S}\sum_{j=1}^{N_i}\alpha_{(i,j)}(1-p_{i,j}) \\
&= \left(\sum_{i=2}^{S}\sum_{j=1}^{N_i}\mathcal{Q}_{\mu_{(i,j)}-1}P_{\mu_{(i,j)}-1}Q_{i,j}(1-p_{i,j})\right) - \left(P_k\sum_{i=2}^{S}\sum_{j=1}^{N_i}\mathcal{Q}_{\mu_{(i,j)}-1}Q_{i,j}(1-p_{i,j})\right) \\
&= \left(\sum_{i=2}^{S}\sum_{j=1}^{N_i}\mathcal{Q}_{\mu_{(i,j)}-1}P_{\mu_{(i,j)}-1}Q_{i,j}(1-p_{i,j})\right) - P_k(1-\mathcal{Q}_k) \\
&= \left(\sum_{i=2}^{S}\sum_{j=1}^{N_i}(\mathcal{Q}_{\mu_{(i,j)}-1}P_{\mu_{(i,j)}-1}Q_{i,j} - \mathcal{Q}_{\mu_{(i,j)}-1}P_{\mu_{(i,j)}-1}Q_{i,j}p_{i,j})\right) - P_k(1-\mathcal{Q}_k) \\
&= \left(\sum_{m=1}^{k}\sum_{\substack{(i,j)\text{ s.t.}\\ \mu_{(i,j)}=m}}(\mathcal{Q}_{m-1}P_{m-1}Q_{i,j} - \mathcal{Q}_{m-1}P_{m-1}Q_{i,j}p_{i,j})\right) - P_k(1-\mathcal{Q}_k) \\
&= \left(\sum_{m=1}^{k}\mathcal{Q}_{m-1}P_{m-1}\sum_{\substack{(i,j)\text{ s.t.}\\ \mu_{(i,j)}=m}}(Q_{i,j} - Q_{i,j}p_{i,j})\right) - P_k(1-\mathcal{Q}_k) \\
&= \left(\sum_{m=1}^{k}\mathcal{Q}_{m-1}P_{m-1}(1-Q_m)\right) - P_k(1-\mathcal{Q}_k) \ .
\end{aligned}$$

The last equality above is established using the same type of reasoning as the one we used to establish Equation (27).

We now combine Inequality (25) with Equation (26):

$$
\begin{aligned}
&\sum_{m=1}^{k} (\mathcal{Q}_m - 1)\, P_{m-1} a_{\sigma(m)} + A \sum_{i=2}^{S} \sum_{j=1}^{N_i} \alpha_{(i,j)} (1 - p_{i,j}) \\
&\geq A \left(\left(\sum_{m=1}^{k} (\mathcal{Q}_m - \mathcal{Q}_{m-1}) P_{m-1} \right) + P_k (1 - \mathcal{Q}_k) \right) + A \left(\left(\sum_{m=1}^{k} \mathcal{Q}_{m-1} P_{m-1} (1 - Q_m) \right) - P_k (1 - \mathcal{Q}_k) \right) \\
&= A \left(\left(\sum_{m=1}^{k} (\mathcal{Q}_m - \mathcal{Q}_{m-1}) P_{m-1} \right) + P_k (1 - \mathcal{Q}_k) + \left(\sum_{m=1}^{k} \mathcal{Q}_{m-1} P_{m-1} (1 - Q_m) \right) - P_k (1 - \mathcal{Q}_k) \right) \\
&= A \left(\left(\sum_{m=1}^{k} (\mathcal{Q}_m - \mathcal{Q}_{m-1}) P_{m-1} \right) + \left(\sum_{m=1}^{k} \mathcal{Q}_{m-1} P_{m-1} (1 - Q_m) \right) \right) \\
&= A \left(\sum_{m=1}^{k} (\mathcal{Q}_m P_{m-1} - \mathcal{Q}_{m-1} P_{m-1} + \mathcal{Q}_{m-1} P_{m-1} - \mathcal{Q}_{m-1} P_{m-1} Q_m) \right) \\
&= 0 \,,
\end{aligned}
$$

because $\mathcal{Q}_{m-1} Q_m = \mathcal{Q}_m$. We have thus established Inequality (24), which concludes the proof. □

D Dominance computation algorithm

DirectDomination(Algorithm 6) computes the direct (non-transitive) dominance relationships between the leaves. It calls TransitiveDomination(Algorithm 7), which computes the transitive dominance relationships.

Algorithm 6: DIRECTDOMINATION($\mathcal{L}$, *NotYetScheduled*)

```
1  (Source, DominatesTrans) = TRANSITIVEDOMINATION(L, NotYetScheduled)
2  for l_first ∈ L do
3      for l_second ∈ {l_first+1, ..., l_L} do
4          DominatesDirect[l_first][l_second] ← FALSE
5  for l_first ∈ L do Check whether l_first dominates directly l_second
6      if (NotYetScheduled[l_first]) then
7          for l_second ∈ L \ {l_first} do
8              if (NotYetScheduled[l_second] and DominatesDirect[l_first][l_second]) then
9                  NoMiddleLeaf ← TRUE
10                 for l_middle ∈ L \ {l_first, l_second} do
11                     if NotYetScheduled[l_middle]
12                         and DominatesDirect[l_middle][l_second]
13                         and DominatesDirect[l_first][l_middle]) then
14                         NoMiddleLeaf ← FALSE
15                 DominatesDirect[l_first][l_second] ← NoMiddleLeaf
16 return DominatesDirect
```

Algorithm 7: TRANSITIVEDOMINATION($\mathcal{L}$, *NotYetScheduled*)

```
for l_i ∈ L do Initialization
    Source[l_i] ← TRUE
for l_first in L do Computation of the dominance relationship
    if (NotYetScheduled[l_first]) then
        for l_second ∈ {l_{first+1}, ..., l_L} do
            if (NotYetScheduled[l_j]) then
                FirstDominatesSecond ← TRUE; SecondDominatesFirst ← TRUE
                for s ∈ S do Loop on all streams
                    if (d^s_{l_first} > d^s_{l_second}) then
                        SecondDominatesFirst ← FALSE
                    else if (d^s_{l_first} < d^s_{l_second}) then
                        FirstDominatesSecond ← FALSE
                if (SecondDominatesFirst and FirstDominatesSecond) then
                    if (p_{l_first} < p_{l_second}) then
                        DominatesTrans[l_second][l_first] ← TRUE
                        DominatesTrans[l_first][l_second] ← FALSE
                        Source_{l_second} ← FALSE
                    else
                        DominatesTrans[l_second][l_first] ← FALSE
                        DominatesTrans[l_first][l_second] ← TRUE
                        Source_{l_first} ← FALSE
                else
                    DominatesTrans[l_first][l_second] ← FirstDominatesSecond
                    DominatesTrans[l_second][l_first] ← SecondDominatesFirst
                    if (FirstDominatesSecond) then
                        Source[l_first] ← FALSE
                    else if (SecondDominatesFirst) then
                        Source[l_second] ← FALSE
return (Source, DominatesTrans)
```

E Proof of Theorem 4

Proof. The problem is obviously in NP: given a schedule, i.e., an ordering of the leaves, one can compute its expected cost in polynomial time, using the method given in Section 4.2. The NP-completeness is obtained by reduction from 2-PARTITION [15]. Let $\mathcal{I}_1$ be an instance from 2-PARTITION: given a set $\{a_1, ..., a_n\}$ and $S = \sum_{i=1}^{n} a_i$, does there exist a subset I such that $\sum_{i \in I} a_i = \frac{S}{2}$? We assume that S is even, otherwise there is no solution. The size of $\mathcal{I}_1$ is $O(n \log M)$, where $M = \max_{1 \leq i \leq n}\{a_i\}$. Without loss of generality, we assume that $M \geq 10$. We construct the following instance $\mathcal{I}_2$ of DNF-Decision:

- We consider a DNF tree with $N = n + 1$ AND nodes AND_i, $1 \leq i \leq n + 1$ and a total of $L = 2n + 1$ leaves.
- The set of streams is $\mathcal{S} = \{A_1, \ldots, A_n, B\}$. The cost of stream $s_i = A_i$ for $i \leq n$ is $c(i) = \frac{1}{2Z}$, where Z is some large constant defined below. The cost of stream $s_{n+1} = B$ is $c(n + 1) = C_0$, where $C_0 \approx \frac{1}{2}$ is a constant defined below.
- Each AND_i node, where $i \leq n$, has a single leaf $l_{i,1}$ which has success probability

$$p_{i,1} = \frac{a_i}{Z} + \beta \frac{a_i^2}{Z^2}$$

  where $\beta \approx \frac{1}{2}$ is a constant defined below, and which requires $d_{l_{i,1}}^{A_i}$ elements of stream A_i. Hence the cost to access all items of leaf $l_{i,1}$ is $d_{l_{i,1}}^{A_i} c(i) = \frac{a_i}{Z}$.
- The last AND node AND_{n+1} has $m_{n+1} = n + 1$ leaves which are specified as follows:
  - Each leaf $l_{n+1,i}$, where $i \leq n$, has success probability $p_{n+1,i} = 1 - \varepsilon$ and requires $d_{l_{n+1,i}}^{A_i} = a_i$ elements of stream A_i. Hence the cost to access all items of leaf $l_{n+1,i}$ is $\frac{a_i}{2Z}$.
  - The last leaf $l_{n+1,n+1}$ has success probability $p_{n+1,n+1} = 1 - \varepsilon$ and requires $d_{l_{n+1,n+1}}^{B} = 1$ element of stream B (at cost $c(n + 1) = C_0$). Constant ε is chosen to be very small, see below. Let $C = \sum_{i=1}^{n} \frac{a_i}{2Z} + C_0 = \frac{S}{2Z} + C_0$: Intuitively, C would be the cost of evaluating node AND_{n+1} when starting with this AND node, and when ε becomes negligible.
- The bound on the expected evaluation cost is $K = C\left(1 - \frac{S^2}{8Z^2}\right) + \frac{1}{9Z^2}$

To finalize the description of $\mathcal{I}_2$, we define the constants as follows:

- $Z = 10\left((n + 1)3^n + n^3\right) M^3$
- $C_0 = \frac{Z}{2Z - S} - \frac{S}{2Z}$, so that $C = \frac{Z}{2Z - S}$
- $\beta = \frac{1 - C}{2C}$
- $\varepsilon = \frac{1}{90(n+1)^2 Z^2}$

The size of $\mathcal{I}_2$ is polynomial in the size of $\mathcal{I}_1$: the greatest value in $\mathcal{I}_2$ is Z and $\log(Z)$ is linear in $(n + \log M)$. Because Z is very large in front of $S \leq nM$, we do have that C, C_0 and β are all close to $\frac{1}{2}$. We only use that these constants are all non-negative, and that $\beta \leq 1$ and $C \leq 1$, in the following derivation, where we bound the expected cost of an arbitrary evaluation of the DNF tree. Then, using this derivation, we will prove that $\mathcal{I}_1$ has a solution I if and only if $\mathcal{I}_2$ does.

Let us start with the cost of an arbitrary evaluation of the DNF tree. Owing to the dominance property stated in Theorem 3, we can assume that the schedule is depth-first. Therefore, a schedule first evaluates n AND nodes in sequence and completely before starting the evaluation of node AND_{n+1}. Then, because ε is very small, we can compute an approximation of the cost by assuming that the schedule terminates after node AND_{n+1}. This is because all its leaves have success probability close to 1. We will bound the difference between this approximation and the actual cost later on.

Let $I = \{\mathrm{AND}_{\sigma(1)}, \mathrm{AND}_{\sigma(2)}, \ldots, \mathrm{AND}_{\sigma(k)}\}$ be the subset, of cardinal k, of AND nodes that are evaluated, in that order, before node AND_{n+1}. Let $\overline{Cost}$ be the approximated cost of the schedule (terminating after completion of node AND_{n+1}). To simplify notations, we let $x_i = a_{\sigma(i)}$ for $1 \leq i \leq k$, and let $q_i = 1 - p_{\sigma(i),1}$ for $i \leq n$. By definition,

$$\overline{Cost} = \sum_{i=1}^{k} \frac{x_i}{Z} \prod_{1 \leq j < i} q_j + \left(C - \sum_{i=1}^{k} \frac{x_i}{2Z} \right) \prod_{1 \leq j \leq k} q_j.$$

Note that the cost of node AND_{n+1} has been reduced from its original value, due to the sharing of the streams whose index is in I. To evaluate $\overline{Cost}$, we start by approximating

$$\prod_{1 \leq j < i} q_j = \prod_{1 \leq j < i} \left(1 - \frac{x_j}{Z} - \beta \frac{x_j^2}{Z^2} \right)$$

Let

$$F_i = 1 - \sum_{j=1}^{i-1} \frac{x_j}{Z} - \beta \sum_{j=1}^{i-1} \frac{x_j^2}{Z^2} + \sum_{1 \leq j_1 < j_2 < i} \frac{x_{j_1} x_{j_2}}{Z^2}.$$

We have

$$\left| \left(\prod_{1 \leq j < i} q_j \right) - F_i \right| \leq \frac{3^n M^3}{Z^3} \tag{28}$$

To see this, we have kept in F_i all terms of the product $\prod_{1 \leq j < i} q_j$ whose denominators include a factor strictly inferior to Z^3. The other terms of the product are bounded (in absolute value) by M^3/Z^3, because $\beta \leq 1$ and $M \leq Z$. There are at most $3^{i-1} \leq 3^n$ such terms. Hence, the desired bound in Equation (28). Letting

$$G = \sum_{i=1}^{k} \frac{x_i}{Z} - \sum_{1 \leq j_1 < j_2 \leq k} \frac{x_{j_1} x_{j_2}}{Z^2},$$

we prove similarly that

$$\left| \left(\sum_{i=1}^{k} \frac{x_i}{Z} \prod_{1 \leq j < i} q_j \right) - G \right| \leq \frac{n 3^n M^3}{Z^3}. \tag{29}$$

Indeed, there are $k \leq n$ terms in the sum, each of them being bounded as before. We deduce from Equations (28) and (29), using $C \leq 1$, that

$$\left| \overline{Cost} - \left(G + (C - \sum_{i=1}^{k} \frac{x_i}{2Z}) F_{k+1} \right) \right| \leq \frac{(n+1) 3^n M^3}{Z^3} \tag{30}$$

Now, we aim at simplifying $H = G + (C - \sum_{i=1}^{k} \frac{x_i}{2Z}) F_{k+1}$ by dropping terms whose denominator is Z^3. We have

$$H = \sum_{i=1}^{k} \frac{x_i}{Z} - \sum_{1 \leq j_1 < j_2 \leq k} \frac{x_{j_1} x_{j_2}}{Z^2} + \left(C - \sum_{i=1}^{k} \frac{x_i}{2Z} \right) \left(1 - \sum_{j=1}^{k} \frac{x_j}{Z} - \beta \sum_{j=1}^{k} \frac{x_j^2}{Z^2} + \sum_{1 \leq j_1 < j_2 \leq k} \frac{x_{j_1} x_{j_2}}{Z^2} \right)$$

Defining

$$\tilde{H} = C + \frac{1-2C}{2Z}\sum_{i=1}^{k} x_i + \frac{1}{2Z^2}\left(\sum_{i=1}^{k} x_i\right)^2 + \frac{C-1}{Z^2}\sum_{1\le j_1<j_2\le k} x_{j_1}x_{j_2} - \frac{\beta C}{Z^2}\sum_{i=1}^{k} x_i^2,$$

we derive (using $\beta \le 1$) that:

$$\left|H - \tilde{H}\right| \le \left|\frac{1}{2Z^3}\left(\sum_{i=1}^{k} x_i\right)\left(\sum_{1\le j_1<j_2\le k} x_{j_1}x_{j_2} + \sum_{i=1}^{k} x_i^2\right)\right|.$$

Hence,

$$\left|H - \tilde{H}\right| \le \frac{n^3M^3}{Z^3} \tag{31}$$

Developing $(\sum_{i=1}^{k} x_i)^2 = \sum_{i=1}^{k} x_i^2 + 2\sum_{1\le j_1<j_2\le k} x_{j_1}x_{j_2}$ in $\tilde{H}$, we obtain

$$\tilde{H} = C + \frac{1-2C}{2Z}\sum_{i=1}^{k} x_i + \frac{C}{Z^2}\sum_{1\le j_1<j_2\le k} x_{j_1}x_{j_2} + \frac{1-2\beta C}{2Z^2}\sum_{i=1}^{k} x_i^2$$

We have chosen the constants C and β so that $\tilde{H}$ can be reduced to

$$\tilde{H} = C + \frac{C}{2Z^2}\left(\left(\frac{S}{2} - \sum_{i=1}^{k} x_i\right)^2 - \frac{S^2}{4}\right) \tag{32}$$

Indeed, we have $\frac{1-2C}{2Z} = \frac{-SC}{2Z^2}$, and $C = 1 - 2C\beta$. Altogether, we derive from Equations (30) to (32) that

$$\left|\overline{Cost} - C\left(1 - \frac{S^2}{8Z^2}\right) - \frac{C}{2Z^2}\left(\frac{S}{2} - \sum_{i=1}^{k} x_i\right)^2\right| \le \frac{\left((n+1)3^n + n^3\right)M^3}{Z^3} = \frac{1}{10Z^2} \tag{33}$$

Finally, we coarsely bound the difference between the actual cost *Cost* of the schedule and the approximated cost $\overline{Cost}$. The actual probability of evaluating the i-th leaf of node AND_{n+1} is $(1-\varepsilon)^i$ so that the error term for that leaf does not exceed $\left(1-(1-\varepsilon)^i\right)\max(\frac{M}{2Z}, C) \le n\varepsilon$. Since there are $n+1$ terms, we get a difference bounded by $n(n+1)\varepsilon$. Next we have neglected the evaluation of the remaining AND nodes after node AND_{n+1}, but this cost is (similarly) bounded by $(n+1)\varepsilon\frac{S}{Z} \le (n+1)\varepsilon$. Altogether, we obtain that

$$\left|Cost - \overline{Cost}\right| \le (n+1)^2\varepsilon = \frac{1}{90Z^2} \tag{34}$$

Combining Equations (33) and (34), we finally derive that

$$\left|Cost - C\left(1 - \frac{S^2}{8Z^2}\right) - \frac{C}{2Z^2}\left(\frac{S}{2} - \sum_{i=1}^{k} x_i\right)^2\right| \le \frac{1}{9Z^2} \tag{35}$$

We now prove that $\mathcal{I}_1$ has a solution I if and only if $\mathcal{I}_2$ does. Suppose first that $\mathcal{I}_1$ has a solution I: $\sum_{i\in I} a_i = \frac{S}{2}$. We evaluate the AND nodes whose indices are in I before evaluating

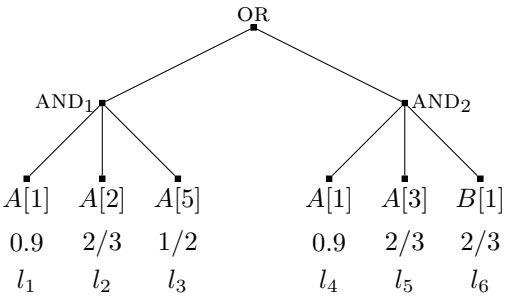


Figure 12: Example DNF tree for which SingleStreamGreedy does not produce an optimal schedule.

node AND_{n+1}, followed by the remaining AND nodes in any order. Let *Cost* be the cost of this evaluation. From Equation (35), we have

$$\left| Cost - C\left(1 - \frac{S^2}{8Z^2}\right)\right| \leq \frac{1}{9Z^2}.$$

Hence, $Cost \leq C\left(1 - \frac{S^2}{8Z^2}\right) + \frac{1}{9Z^2} = K$, thereby providing a solution to $\mathcal{I}_2$.

Suppose now that $\mathcal{I}_2$ has a solution whose cost is $Cost \leq K$, and let I denote the (index) set of AND nodes that are evaluated before node AND_{n+1}. If (by contradiction) we have $\sum_{i\in I} a_i \neq \frac{S}{2}$, then $\left(\frac{S}{2} - \sum_{i=1}^{k} x_i\right)^2 \geq 1$, and Equation (35) shows that

$$Cost \geq C\left(1 - \frac{S^2}{8Z^2}\right) + \frac{C}{2Z^2} - \frac{1}{9Z^2} = K + \frac{9C-4}{9Z^2}.$$

Since $9C - 4 = \frac{Z+4S}{2Z-S} > 0$, then $Cost > K$, which is a contradiction. Therefore $\sum_{i\in I} a_i = \frac{S}{2}$, and $\mathcal{I}_1$ has a solution. This concludes the proof.

It is interesting to point out that instance $\mathcal{I}_2$ is constructed so that the ordering of the leaves inside each AND node has no importance. In fact, only the last AND node has more than one leaf, and because its leaves have all very high success probability, their ordering does not matter. This shows that the combinatorial difficulty of the DNF-Decision problem already lies in deciding the ordering of the AND nodes. □

F Counter-example for Algorithm 2 on DNF trees

In this section we provide a counterexample to show that for *shared* queries SingleStream-Greedy, which is optimal to evaluate an AND tree, cannot be used to optimally evaluate a DNF tree.

We consider the example of Figure 12 where both streams have a cost of 1. There are two possible orders for evaluating the AND nodes. We consider both of them and explicit the behavior of SingleStreamGreedy on each of the AND's.

- AND_1 then AND_2. Because of Proposition 2, the leaves of AND_1 are always evaluated in the order l_1, l_2, l_3. The cost of evaluation of AND_1 is then $\alpha_1 = 1+0.9\times(1+2/3\times 3) = 3.7$. We then move to the evaluation of AND_2. SingleStreamGreedy first schedules leaf l_5 whose cost is null. We then have to compare the ratios for leaves l_5 and l_6:

  - l_5. The first element of A was acquired for the evaluation of leaf l_1. The second element of A needs to be acquired for l_2 only if leaf l_2 was not evaluated and leaf l_4 evaluated to TRUE, which happens with probability $(1-0.9)\times 0.9 = 0.09$. The third element of A needs to be acquired only if leaf l_3 was not evaluated and leaf l_4 evaluated to TRUE, which happens with probability $(1-0.9\times 2/3)\times 0.9 = 0.36$. Therefore, the evaluation cost of l_5 is $0+0.09\times 1+0.36\times 1 = 0.45$. The ratio for leaf l_5 is thus $\frac{0.45}{1-2/3} = 1.35$.
  - l_6. The ratio for l_6 is $\frac{1}{1-2/3} = 3$.

  Therefore, SingleStreamGreedy schedules l_5 and then l_6 for the overall cost:

$$3.7 + 0 + 0.45 + (1 - 0.9 \times 2/3 \times 1/2) \times 0.9 \times 2/3 = 4.57.$$

- AND_2 then AND_1. We first consider the ratios for the three leaves:
  - The ratio for l_4 is $\frac{1}{1-0.9} = 10$.
  - The ratio for l_5 is $\frac{1+0.9\times 2}{1-0.9\times 2/3} = 7$.
  - The ratio for l_6 is $\frac{1}{1-2/3} = 3$.

  Therefore the first scheduled leaf if l_6. Then the overall schedule is l_6, l_4, l_5, l_1, l_2, l_3. We compute the evaluation cost of each leaf.
  - l_6. Its cost is 1.
  - l_4. Its cost is $2/3 \times 1$.
  - l_5. Its cost is $2/3 \times 0.9 \times 2 = 1.2$.
  - l_1. Its cost is $(1-2/3)\times 1$.
  - l_2. Its costs is $(1-2/3\times 0.9)\times 0.9 = 0.36$.
  - l_3. Its costs is $(1-2/3\times 0.9)\times 1 + (1-2/3\times 0.9\times 2/3)\times 0.9\times 2/3\times 2 = 0.96$.

  The overall cost is thus 4.52.

We now consider the schedule l_4, l_5, l_6, l_1, l_2, l_3. We compute the evaluation cost of each leaf.

- The cost of l_4 is 1.
- The cost of l_5 is $0.9 \times 2 = 1.8$.
- The cost of l_6 is $0.9 \times 2/3 \times 1 = 0.6$.
- The cost of l_1 is 0.
- The cost of l_2 is $(1-0.9)\times 0.9 = 0.09$.
- The cost of l_3 is $(1-0.9)\times 0.9\times 2/3 + (1-0.9\times 2/3\times 2/3)\times 0.9\times 2/3\times 2 = 0.78$.

The overall cost of this schedule is thus 4.27. The intuitive explanation is as follows: for AND_2 alone, the best evaluation order is l_6, l_4, l_5 (and this is the order chosen by SingleStreamGreedy). However, because of the re-use of some data items of stream A in AND_1, the optimal order for the whole DNF tree is not the same! This leads to a enormous combinatorial search space for the optimal ordering, which corroborates the hardness result (NP-completeness stated in Theorem 4) of the evaluation of *shared* DNF query trees.

G Dominant Linear Strategy Counter-Example

A more general notion than a schedule, called a *strategy*, is described in [8]. Although it may seem counter-intuitive, the processing of a query does not have to follow a defined ordering of the leaves. Instead, a strategy is a decision tree in which the next leaf to be evaluated is chosen based on the truth value of the leaves that have been evaluated previously. A *schedule*, as defined in this work, is a particular kind of strategy, termed a "linear strategy" in [8]. Therein, the authors

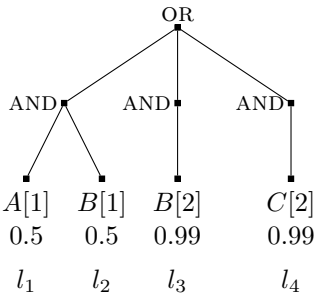


Figure 13: Example DNF tree for which the best schedule has larger cost than a non-linear strategy.

prove that for some problem instances the best linear strategy can be far from being the optimal strategy. Although interesting from a theoretical standpoint, a practical drawback of a non-linear strategy is that the size of its description is exponential in the number of tree leaves. Instead, a linear strategy, or schedule, is simply an ordering of the leaves, with a description size linear in the number of tree leaves. This severe drawback explains why we have not considered non-linear strategies in this work.

However, from a theoretical point of view, it is interesting to ask the following question: while linear strategies are dominant (among all possible strategies) for DNF trees in the *read-once* case [8], is it still the case in the *shared* case? We show that the answer is negative by building a counter-example.

Consider three streams, A, B, and C, with per data item costs $c(A) = 1$, $c(B) = 1.1$, and $c(C) = 1$. Consider the query tree in Figure 13, where for each leaf is indicated the success probability, the stream needed, and the number of data items required from that stream. We first compute the best schedule (i.e., leaf ordering). The cost of schedule l_1, l_2, l_3, l_4 is

$$\begin{aligned} c(A) + p_1(c(B) + (1-p_2)c(B)) + (1-p_1)(2c(B)) \\ +(1-p_1p_2)(1-p_3)(2c(C)) = 1.95 < 2 \end{aligned}$$

The cost of any schedule starting with l_3 or l_4 is at least 2. The cost of schedule l_1, l_2, l_4, l_3 is larger than

$$c(A) + p_1(c(B) + (1-p_1p_2)(2c(C)) = 2.15 > 2.$$

Finally, if we start with the first AND node, it is always better to start with leaf l_1 whose cost is 0. Altogether, the best schedule is l_1, l_2, l_3, l_4, of cost 1.95.

Now, consider the non-linear strategy that evaluates l_1 first and then:

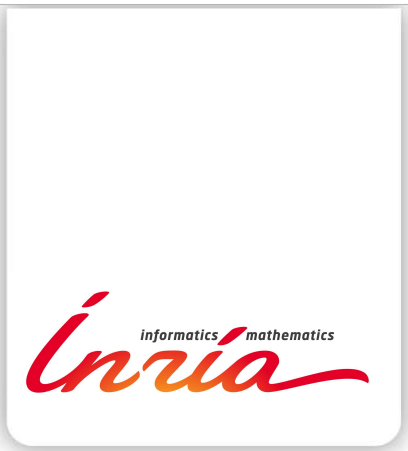
- if l_1 evaluates to TRUE, proceeds with l_2, l_3, and l_4, just as in the optimal schedule;
- if l_1 evaluates to FALSE, proceeds with l_4, l_3, and l_2.

The cost of this strategy is

$$\begin{aligned} c(A)+ \\ p_1[(c(B) + (1-p_2)c(B)) + (1-p_2)(1-p_3)(2c(C))] \\ +(1-p_1)[2c(C)) + (1-p_4)(2c(B))] = 1.851, \end{aligned}$$

which is lower than that of the best schedule.

Determining the optimal non-linear strategy for a DNF tree in the *shared* model is an open problem. Unless some structural property of this strategy can be proven, the space required to describe this optimal non-linear strategy is unknown (and likely exponential).



RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399