

A Generic Flow Algorithm for Shared Filter Ordering Problems

Zhen Liu, Srinivasan Parthasarathy, Anand Ranganathan, Hao Yang
IBM T.J.Watson Research Center
19 Skyline Dr., Hawthorne, NY 10532, USA
Email: {zhenl, spartha, arangana, haoyang}@us.ibm.com

ABSTRACT

We consider a fundamental flow maximization problem that arises during the evaluation of multiple overlapping queries defined on a data stream, in a heterogenous parallel environment. Each query is a conjunction of boolean filters, and each filter could be shared across multiple queries. We are required to design an evaluation plan that evaluates filters against stream items in order to determine the set of queries satisfied by each item. The evaluation plan specifies for each item: (i) the subset of filters evaluated for this item and the order of their evaluations, and (ii) the processor on which each filter evaluation occurs. Our goal is to design an evaluation plan which maximizes the total throughput (flow) of the stream handled by the plan, without violating the processor capacities.

Filter ordering has received extensive attention in single-processor settings, with the objective of minimizing the total cost of filter evaluations: in particular, efficient (approximation) algorithms are known for various important versions of min-cost filter ordering. Min-cost filter ordering problem for a single processor is a special case of our flow maximization for parallel processors. Our main contribution in this work is a generic flow-maximization algorithm, which assumes the availability of a min-cost filter ordering algorithm for a single processor, and uses this to iteratively construct a solution to the flow-maximization problem for heterogenous parallel processors. We show that the approximation ratio of our flow-maximization strategy is essentially the same as that of the underlying min-cost filter ordering algorithm. Our result, along with existing results on min-cost filter ordering, enables the optimization of several important versions of filter ordering in parallel environments.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

PODS'08, June 9–12, 2008, Vancouver, BC, Canada.

Copyright 2008 ACM 978-1-60558-108-8/08/06 ...\$5.00.

Categories and Subject Descriptors

F.2.2 Nonnumerical Algorithms and Problems

General Terms

Algorithms, Performance, Theory

Keywords

Shared filter ordering, Flow maximization, Query optimization, Parallel

1. INTRODUCTION

We consider a fundamental flow maximization problem that arises in the evaluation of multiple overlapping queries defined on a data stream. In our problem, we are given (i) a collection of commutative boolean filters; (ii) a collection of queries, each of which is a conjunction of a subset of filters; and (iii) a collection of heterogenous parallel processors on which the filters could be evaluated along with a capacity value for each processor. Evaluating a specific filter on different processors could, in general, induce different processing loads on them. Evaluating a specific filter on a stream item results in a *true* value (if the item *satisfies* the filter) or *false* value (otherwise). An item satisfies a query if it satisfies all of its filters.

An *evaluation plan* identifies the set of queries satisfied by each stream item; it consists of an *adaptive ordering* which incrementally selects the filters that are evaluated for each stream item, as well a *schedule* which decides the processor on which each filter evaluation is performed. Our goal is to devise an evaluation plan which maximizes the overall throughput or flow (in stream items per second) that is handled by the system, subject to the constraint that the total load on each processor does not exceed its capacity. The choice of the evaluation plan is simultaneously influenced by many factors including: filter selectivities (i.e., the probability that a stream item satisfies a given filter), correlation between filter selectivities, filter popularities (i.e., the number of queries that contain a given filter), and the filter-processor load values (i.e., time taken to evaluate a given filter on a given processor).

Filter ordering problems command a long history in database and stream computing literature [15, 18, 13, 21, 19, 7, 17], and have been at the focus of renewed attention due to their applications in web services [4, 9, 12], high-throughput data stream computing [1, 2], sensor networks [8] and multimedia stream analysis and classification [19]. Much of the existing work [15, 18, 13, 21, 2, 19] deals with a *cost-minimization*

problem that arises in the context of a *single-processor* environment. In the *min-cost filter ordering problem*, we are given a fixed cost for evaluating each filter, and our goal is to adaptively order the filter evaluations for each stream item in order to determine all the queries satisfied by the item, while minimizing the total expected cost of evaluation. Min-cost filter ordering for the special case of a single query and filters whose selectivities are mutually independent has been considered earlier [15, 18, 13] and it is well-known that a natural greedy strategy is provably optimal in this case. Babu *et al.* [2] studied the case of a single query with arbitrarily correlated filters; they showed that min-cost filter ordering is NP-Complete in this setting and presented a 4-approximation algorithm for this problem. Min-cost filter ordering in the setting of multiple overlapping queries and mutually independent filters was considered by Munagala *et al.* [21], who showed the problem to be NP-Complete and presented a $O(\log^2(n)\log(m))$ -approximation algorithm for the problem, where n and m are the number of filters and queries respectively. Liu *et al.* [19] present an improved $1 + \log(n) + \log(m)$ -approximation algorithm for the same setting as [21].

In contrast to min-cost filter ordering which deals with a single-processor setting, our concern in this work is flow-maximization which deals with the joint problem of filter ordering and scheduling in a heterogeneous parallel-processor environment. Indeed, min-cost filter ordering is easily seen to be a special case of flow-maximization, when the latter is restricted to a single-processor and the processing load of a filter is equal to its cost. Thus, we observe that designing a θ -approximation algorithm for the flow-maximization problem also yields a θ -approximation for the min-cost filter ordering problem. *Our central contribution in this work is a surprising result which shows that the converse is also true.*

We develop a generic algorithm for flow-maximization called Virtual Cost Vectors (VCV). VCV takes as input an instance of flow-maximization, as well as a polynomial-time γ -approximation algorithm for min-cost filter ordering. Using the latter as a subroutine, VCV builds (essentially) a γ -approximate solution to flow maximization. In particular, given parameters $\tau > 0$ and $\phi \in (0, 1)$, with probability ϕ , VCV is guaranteed to produce a feasible solution to flow maximization that is at most a factor of $\gamma \cdot (1 + \tau)$ from the optimal flow value. The runtime of VCV is guaranteed to be polynomial in the number of filters, queries, and processors for any fixed values of τ and ϕ . Thus, consider the flow maximization problem with (i) single query and independent filter selectivities, (ii) single query and arbitrarily correlated filter selectivities, and (iii) multiple overlapping queries with independent filter selectivities. VCV in conjunction with the work of [15, 18, 13], [2], and [19], yields with high-probability, a $(1 + \tau)$ -approximation, $4 \cdot (1 + \tau)$ -approximation, and $(1 + \log(n) + \log(m)) \cdot (1 + \tau)$ -approximation for the above problems respectively.

The VCV algorithm computes *virtual* cost-functions $y_\ell(F_i, M_k)$ which specify a cost for evaluating filter F_i on processor M_k . Each cost-function y_ℓ , along with the min-cost filter ordering algorithm, represents a specific adaptive ordering and scheduling strategy, and thus acts as an atomic component of the solution to the flow maximization. Indeed, for a given cost-function y_ℓ , the scheduling strategy is utterly simple: whenever we evaluate filter F_i , we always do so on a processor M_k such that $y_\ell(F_i, M_k)$ is minimum. This leaves us

only with the problem of filter ordering which, for a given cost-function y_ℓ , is solved by the min-cost filter ordering algorithm. The final solution output by VCV simply consists of a collection of cost functions, and the fraction of the input flow that is routed through each cost function.

At the heart of VCV lies the procedure for iteratively computing the virtual cost-functions. This procedure is inspired by the seminal work of Plotkin, Shmoys and Tardos [22] (and subsequently built upon by others [11, 25]) which provides for a fast approximation algorithm based on primal-dual techniques, to solve certain classes of linear programs. However, a major distinction between our setting and the above mentioned one is that the latter assumes that the constraint matrix for the linear program is given as part of the input; in our case, the constraint matrix is of exponential size and is not part of the input. We overcome this through the use of a carefully constructed sampling procedure, which estimates the relevant portions of the constraint matrix alongside the iteratively computed cost-functions. Thus, the VCV algorithm and its analysis is derived through a careful fusion of primal-dual techniques, sampling procedures, as well as other problem specific ideas.

1.1 Related Work

The work of Kodialam [17] and Condon *et al.* [7] is most similar in spirit to our work. Kodialam [17] considered the flow maximization problem in the context of a single query with filters whose selectivities are mutually independent under the assumption that each filter is assigned to exactly a single processor (i.e., dedicated filter placements); he presented an optimal flow algorithm for this scenario whose runtime is $O(n^3 \log n)$. The recent work of Condon *et al.* [7] improves upon Kodialam's result by providing an $O(n^2)$ algorithm for the same problem. Our work does not directly challenge the results of [17] and [7] – in their setting, our algorithm produces a $(1 + \tau)$ -approximation with probability ϕ , where $\tau > 0$ and $\phi \in (0, 1)$ are input parameters which affect the runtime of our algorithm. However, our algorithm solves a substantially more general problem than [17] and [7]. The utility of our algorithm stems from the fact that it makes no assumption about either the combinatorial structure of the query-filter bipartite graph, or the correlation between filter selectivities, but instead relies on the underlying min-cost filter ordering algorithm to handle these issues.

As noted in Section 1, various versions of the min-cost filter ordering problem have received substantial attention in the literature [15, 18, 13, 21, 2, 19]. This problem, and hence our flow maximization problem, both generalize the classical minimum set-cover problem [24]. A variant of the set-cover problem, namely, min-sum set cover possesses a similar combinatorial flavor as single-query min-cost filter ordering [2]; this problem is known to be NP-Complete and has received much attention in the field of approximation algorithms [3, 6, 10, 20]. In terms of single query min-cost filter ordering, the min-sum set cover problem can be viewed as follows: we are given a collection of items, the subset of filters satisfied by each item, and the cost of each filter. We need to determine a fixed order of filters with which to evaluate all items; this is subject to the constraint that an item can be dropped after the evaluation of first filter in the order that is not satisfied by the item; our goal is to determine an order which minimizes the total cost over

all evaluations of the items. Single query min-cost filter ordering has also been an object of study in other areas such as fault detection [23] and machine learning [16].

2. PRELIMINARIES

2.1 Problem Formulation

An instance of the flow maximization problem consists of a set of queries \mathcal{Q} , and a set of commutative filters \mathcal{F} which process the input data stream, and a set of processors \mathcal{M} on which the filter evaluations occur. Each filter $F_i \in \mathcal{F}$ takes a stream item t as input and returns either *true* or *false* as output. If filter F_i returns *true* for item t , we say that t satisfies F_i . A query $q \in \mathcal{Q}$ is a conjunction of a subset of filters $F(q) \subseteq \mathcal{F}$; if a stream item t satisfies all the filters in $F(q)$, we say that t satisfies query q . If we decide to invoke filter F_i for a stream item t , then we also need to assign a processor $M_k \in \mathcal{M}$ on which F_i will be evaluated for item t .

The *evaluation plan* determines, for each item t in the input stream, an *adaptive ordering and schedule* of filters $F_{\sigma(1)}, F_{\sigma(2)}, \dots, F_{\sigma(r)}$ such that the following properties hold: **(P1)**: If t satisfies query q , then all the filters in $F(q)$ appear in the adaptive ordering. **(P2)**: If t does not satisfy q , then at least one filter in $F(q)$ which is not satisfied by t appears in the adaptive ordering. For each filter F which is selected by the adaptive ordering, the schedule determines the specific processor $M_k \in \mathcal{M}$ on which filter F is evaluated. The phrase *adaptive ordering* underscores the fact that, for a given stream item t , the choice of the $i + 1^{\text{st}}$ filter $F_{\sigma(i+1)}$ in the ordering depends on the outcome of the previous i filter evaluations. Further, there could be internal random choices within a plan introduced for the sake of load balancing, which also contributes to the adaptivity. The *schedule* determines the processor on which each filter evaluation occurs. We emphasize that, in general, a single fixed plan could adaptively construct different filter orderings and schedules for different stream items.

Let $L(F_i, M_k)$ denote the processing load incurred by processor M_k when a stream item t invokes filter F_i on M_k .¹ Consider an item t which is chosen uniformly at random from the input stream. Given a plan \mathcal{P} , let $A(\mathcal{P}, F_i, M_k)$ denote the unconditional probability of the following event: “plan \mathcal{P} invokes filter F_i on processor M_k for item t ”. Let $r(\mathcal{P})$ (in items/sec) be the rate at which stream items are processed by plan \mathcal{P} . In order for the plan to be *stable*, the total *load* on each processor must not exceed its capacity: i.e., **(P3)** $\forall M_k \in \mathcal{M} : r(\mathcal{P}) \sum_{F_i \in \mathcal{F}} A(\mathcal{P}, F_i, M_k) \cdot L(F_i, M_k) \leq b(M_k)$; here, $b(M_k)$ denotes the capacity of processor M_k . For the purpose of bounding the runtime of our algorithm, we will assume that the processor capacities are polynomially bounded: in particular, we will assume that $\min_{M_k \in \mathcal{M}} b(M_k) = 1$, and $\max_{M_k \in \mathcal{M}} b(M_k) \leq n^{\lambda_0}$, where n is the number of filters, and λ_0 is a fixed constant.

DEFINITION 1. *The Flow Maximization Problem is defined as the problem of computing an evaluation plan \mathcal{P}*

¹More generally, the processing load could be a random variable in which case we let $L(F_i, M_k)$ denote the expected processing-load incurred by filter F_i on M_k . All our results continue to hold in this case. The $L(\cdot, \cdot)$ values could be time varying characteristics of the input stream; we always refer to their current values.

which satisfies properties **(P1)**, **(P2)** and **(P3)** such that the total input flow $r(\mathcal{P})$ handled by the plan is maximized.

Since the solution to the min-cost filter ordering problem acts as a crucial subroutine in our solution to the flow maximization problem, we provide a formal definition of the former problem as well. Suppose we are given a cost function y which specifies costs² $y(F_i, M_k)$ for evaluating filter F_i on processor M_k .

DEFINITION 2. *The Min-cost Filter Ordering Problem is defined as the problem of computing an evaluation plan \mathcal{P} which satisfies properties **(P1)** and **(P2)** such that the total expected cost $\sum_{F_i \in \mathcal{F}} \sum_{M_k \in \mathcal{M}} A(\mathcal{P}, F_i, M_k) y(F_i, M_k)$ incurred by the plan is minimized.*

2.2 Evaluation Tree Formulation

We now introduce the notion of *evaluation trees* which will set the stage for our subsequent discussions. An evaluation tree is an atomic component of the evaluation plan and represents a specific adaptive ordering and scheduling of filters. Formally, the evaluation tree is a binary tree³ whose nodes are filter-processor pairs. The root of the tree (F, M) denotes the first filter F which will be evaluated when a stream item needs to be processed, and also the processor M on which F will be evaluated. The root’s *left* child (F_{left}, M_{left}) denotes the second filter-processor pair that will be invoked if the filter F is true, while the *right* child (F_{right}, M_{right}) denotes the second filter-processor pair that will be invoked if the filter F is false. This relationship between a node and its left and right child applies recursively to each internal node within the tree. Traversing the evaluation tree from the root towards one of its leaves corresponds to a particular sequence of events (true/false results, filter selection, and scheduling) which occurred when this tree was used for evaluating a specific stream item. Since each evaluation tree represents a specific adaptive ordering and scheduling of filters, it clearly needs to satisfy properties **(P1)** and **(P2)**.

The flow maximization problem can now be succinctly viewed as a problem of (i) selecting a collection of evaluation trees, and (ii) deciding the fraction of stream items processed by each evaluation tree such that the capacity constraints of the processors are not violated, and the total rate of stream items handled by all the trees together is maximized. Specifically, given an evaluation tree T , with a slight abuse of notation, we will let $A(T, F, M)$ denote the probability that the evaluation tree will invoke filter F on processor M when used to evaluate a randomly chosen stream item; let $f(T)$ (in items/sec) be the rate at which stream items are processed by tree T . Let \mathcal{D} denote the set of all evaluation trees. The flow maximization problem seeks a solution to the following linear program.

$$\max \sum_{T \in \mathcal{D}} f(T) \tag{1}$$

$$\forall M_k, \sum_{F_i} L(F_i, M_k) \sum_{T \in \mathcal{D}} f(T) \cdot A(T, F_i, M_k) \leq b(M_k) \tag{2}$$

²Costs are virtual and need not have a physical interpretation

³Nodes in the tree have either zero or two children

We also note that evaluation trees provide a convenient way of viewing the min-cost filter ordering problem. Given a cost function y , the min-cost filter ordering problem simply asks for a least cost evaluation tree T : i.e., find $\arg \min_{T \in \mathcal{D}} \sum_{F_i \in \mathcal{F}} \sum_{M_k \in \mathcal{M}} A(T, F_i, M_k) y(F_i, M_k)$.

3. OUR APPROACH IN BRIEF

We now present four observations which highlight the key elements of our approach. First, note that the flow maximization LP ((1) and (2)) has an exponential number of variables, since the number of evaluation trees is exponential in the number of filters and processors. However, the flow maximization LP is a *packing* LP.⁴ The primal-dual framework of [22, 11] provides fast combinatorial algorithms for the approximate solution of such packing LPs. These algorithms update the dual variables iteratively, and each update involves choosing the *column of least length* from the primal LP's coefficient matrix and then modifying the primal variables which contain positive coefficients in this column; the update method also simultaneously modifies the dual variables. Here, the length of a column is defined as its dot product with the current dual solution. The only assumption made in this framework is that an efficient (poly-time) procedure exists for choosing the least length column, which suffices to guarantee that their algorithm will terminate in poly-time. These facts leads to our first observation: in the context of our flow maximization problem, the task of choosing a least length column corresponds exactly to the task of choosing the least cost evaluation tree T , where the cost-function is the current dual-solution! Thus, a solution to the min-cost filter ordering problem provides us with an opening for solving the flow maximization problem.

The min-cost filter ordering problem is known to be NP-Complete [21] with the exception of the special case that has only a single query and filters whose selectivities are mutually independent [15, 18, 13] - a simple greedy strategy is provably optimal in this case. However, as noted in Section 1, efficient approximation algorithms are known for other important and non-trivial special cases of min-cost filter ordering. For instance, for the special case of a single query but arbitrarily correlated filter selectivities, Babu *et al.* [20, 2] developed a 4-approximation algorithm; for the case of multiple overlapping queries with mutually independent filter selectivities, Munagala *et al.* [21] developed an $O(\log^2(n) \log(m))$ approximation algorithm, where n and m are the number of filters and queries respectively. The recent work of Liu *et al.* [19] presents an improved $1 + \log(n) + \log(m)$ -approximation algorithm for the same setting as [21]. These facts motivate our second observation: given a γ -approximation algorithm for a special case of the min-cost filter ordering problem, we can utilize this as a subroutine to construct a solution to the corresponding special case of the flow maximization problem. The factor γ eventually manifests itself in the approximation guarantee which we establish for our solution to the flow maximization problem. Thus, consider the following *special cases of the flow maximization problem*: (i) a single query with mutually independent filters, (ii) a single query with arbitrarily correlated filters, and (iii) multiple overlapping queries with mutually independent queries. Our work yields $1 + \tau$,

⁴Recall that a packing LP is of the form: $\max c^T \cdot x$ subject to $Ax \leq B$, where x , A , and c have non-negative coefficients.

$4 \cdot (1 + \tau)$, and $(1 + \log(n) + \log(m)) \cdot (1 + \tau)$ -approximation algorithms for these respective special cases of flow maximization. Here, $\tau > 0$ is a parameter that can be chosen arbitrarily to tradeoff the approximation ratio of our algorithm at the expense of its runtime.

The size of an evaluation tree is generally exponential in the number of filters and processors. Thus an explicit representation of a solution to the flow maximization problem using evaluation trees is problematic. However, an algorithm for the min-cost filter ordering problem, along with a specific cost function, provides an implicit representation for a *single* evaluation tree⁵ which lets us circumvent the representation problem. This leads us to our third observation: a solution to the flow maximization problem can be efficiently represented through a collection of (polynomial number of) cost functions, an associated min-cost filter ordering algorithm, and by specifying the fraction of the input flow that is *routed* using each cost function. This observation is also the reasoning behind the name of our algorithm: the acronym VCV stands for *Virtual Cost Vectors*.

Consider a specific cost vector y and the associated min-cost filter ordering algorithm, which provides an implicit representation for some evaluation tree T . While y is a succinct representation for T , in general no closed form expression for $A(T, F_j, M_k)$ - the probability that tree T evaluates filter F_j on processor M_k - can be derived from y , and hence $A(T, F_j, M_k)$ cannot be computed analytically. The $A(T, \cdot, \cdot)$ -values are however crucial for the cost update rule between successive iterations of our algorithm, as well as for determining the split up of the input flow between various cost-vectors. This leads to our fourth observation: the $A(T, \cdot, \cdot)$ -values can be computed upto the required degree of accuracy through sampling, by generating a polynomial-number of stream inputs and executing the min-cost filter ordering algorithm over these inputs. A potential glitch here is that if an $A(T, \cdot, \cdot)$ -value is negligible (e.g., inverse exponential), then computing it to any reasonable degree of accuracy requires too many (exponential) samples. But in this scenario, we can exploit the fact that the impact of this $A(T, \cdot, \cdot)$ -value on the filter evaluation cost is negligible; hence, we can *round up* estimates of $A(T, \cdot, \cdot)$ -values that are negligibly low to a suitably chosen threshold, without significantly altering the eventual approximation ratio of our algorithm. In a nutshell, our algorithm and its analysis is thus derived through a careful concoction of primal-dual and sampling techniques.

4. VCV ALGORITHM

We now present the VCV algorithm. For ease of exposition, in this section and in Section 5, we will assume dedicated filter placements: i.e., each filter is placed on exactly one processor, and each processor contains a single filter. This assumption altogether eliminates the need for scheduling in the flow maximization problem. In Section 6, we will describe how to extend the algorithm when the assignment of filters to processors is arbitrary (i.e., each filter could be present in an arbitrary subset of processors, and the load incurred by a specific filter on a processor is given as part of the input). Throughout Sections 4 and 5, we will assume

⁵More generally, if the min-cost filter ordering algorithm is randomized, it provides an implicit representation for a distribution over evaluation trees.

the following notation: index j represents a filter and its corresponding processor, index ℓ represents a specific iteration of the VCV algorithm, $b(j)$ represents the capacity of filter (or processor) j in stream items per second (i.e., if the expected number of stream items per second which evaluate filter j is $\geq b(j)$, then the solution is infeasible), $A(T, j)$ is the probability of evaluation tree T invoking filter j , $\tilde{A}(T, j)$ is our estimate of $A(T, j)$ obtained through our sampling subroutine, and $y_\ell(j)$ represents the (dual) cost of filter j during iteration ℓ .

Recall that VCV is a (τ, ϕ) -approximation scheme: given a min-cost filter ordering algorithm with an approximation ratio γ , VCV produces a solution to the flow maximization problem such that, with probability ϕ , the approximation ratio of the solution is at most $\gamma(1 + \tau)$. In the description of VCV (Algorithm 1), n denotes the number of filters, and δ and ρ_1 are parameters whose values are set as follows:

$$\delta = (1 + \epsilon) \cdot ((1 + \epsilon)n)^{-\frac{1}{\epsilon}} \quad (3)$$

$$\rho_1 = 1 + \frac{2}{n^\beta} \quad (4)$$

The parameters β and ϵ are suitably chosen constants whose values will be specified in Lemma 9 and in the proof of Theorem 13 respectively.

We are now ready to describe VCV. VCV (Algorithm 1) starts by initializing the dual costs of each filter (Step 2). The dual-cost vector y_ℓ along with the min-cost filter ordering algorithm implicitly represents the evaluation tree \mathcal{T}_ℓ chosen during iteration ℓ . Within a specific iteration ℓ , for each filter j , the subroutine **Sample-And-Estimate** estimates the probability that \mathcal{T}_ℓ invokes filter j (Step 5). These estimates are used to determine the bottleneck filter j_ℓ^{\min} (Step 6): the term bottleneck emphasizes the fact that if we gradually increase the flow routed through \mathcal{T}_ℓ , then j_ℓ^{\min} would be the first filter to be saturated (under the ideal sampling assumption that the estimates $\tilde{A}(\cdot, \cdot)$ are precisely equal to $A(\cdot, \cdot)$; further, since we assume dedicated filter placements, a filter being saturated is equivalent to its corresponding processor being saturated). The bottleneck rate $r(\mathcal{T}_\ell)$ is the flow value at which the bottleneck filter saturates (Step 7). The estimates $\tilde{A}(\cdot, \cdot)$ are then utilized in the exponential cost-update rule for calculating the updated filter costs (Step 8). Observe that the cost increase of a filter j is *inversely* proportional to the flow-value $\frac{b(j)}{A(\mathcal{T}_\ell, b(j))}$ at which this filter would have saturated under the evaluation tree \mathcal{T}_ℓ . Throughout the course of its execution, VCV keeps track of the dual-value Q_ℓ (Steps 3 and 9): the algorithm terminates when Q_ℓ equals or exceeds one (Steps 4 and 11). The final flow-values $f(\cdot)$ routed through the evaluation trees are scaled down values of the corresponding bottleneck rates $r(\cdot)$ (Step 12).

In order to complete the description of VCV, we now present the details of **Sample-And-Estimate** (Algorithm 2) which is invoked in Step 5 of VCV. In this subroutine, h is a parameter whose value is set as $h = \omega + 3\beta + 3$; ω is a parameter whose value is determined in Lemma 11. Given a cost-vector y_ℓ , for each filter j , **Sample-And-Estimate** returns an estimate $\tilde{A}(\mathcal{T}_\ell, j)$, which is an estimate of the probability that the decision-tree \mathcal{T}_ℓ invokes filter j . **Sample-And-Estimate** creates n^h samples of the stream items and runs the min-cost filter ordering algorithm on each of these items. It then computes the sample averages $\tilde{A}(\cdot, \cdot)$ in Line

Algorithm 1 VCV

- 1: Initialize iteration count: $\ell \leftarrow 1$
 - 2: Create the first cost vector: $\forall filters\ j, y_\ell(j) = \frac{\delta}{b(j)}$
 - 3: Compute dual-value: $Q_\ell = \sum_j y_\ell(j) \cdot b(j)$
 - 4: **while** $Q(y_\ell) < 1$ **do**
 - 5: For all filters j , **Sample-And-Estimate** $\tilde{A}(\mathcal{T}_\ell, j)$
 - 6: Detect bottleneck filter: $j_\ell^{\min} = \arg \min_j \frac{b(j)}{\tilde{A}(\mathcal{T}_\ell, j)}$
 - 7: Set the bottleneck rate: $r(\mathcal{T}_\ell) = \frac{b(j_\ell^{\min})}{\tilde{A}(\mathcal{T}_\ell, j_\ell^{\min})}$
 - 8: Compute new cost-vector:
 $\forall filters\ j, y_{\ell+1}(j) = y_\ell(j) \cdot \left(1 + \epsilon \cdot \frac{b(j_\ell^{\min})\tilde{A}(\mathcal{T}_\ell, j)}{b(j)\tilde{A}(\mathcal{T}_\ell, j_\ell^{\min})}\right)$
 - 9: Compute new dual-value: $Q(y_{\ell+1}) = \sum_j y_{\ell+1}(j) \cdot b(j)$
 - 10: Increment ℓ : $\ell \leftarrow \ell + 1$
 - 11: **end while**
 - 12: Compute final flows: $f(\mathcal{T}_\ell) = \frac{r(\mathcal{T}_\ell)}{\rho_1 \cdot \log_{1+\epsilon} \frac{1+\epsilon}{\delta}}$
 - 13: **return** all cost vectors y_ℓ and associated flows $f(\mathcal{T}_\ell)$
-

10. If an estimate $\tilde{A}(\cdot, \cdot)$ is less than a threshold $\frac{1}{n^\omega}$, then the value of $\tilde{A}(\cdot, \cdot)$ is rounded up to $\frac{1}{n^\omega}$. This completes the description of **Sample-and-Estimate** and VCV.

Algorithm 2 Sample-And-Estimate

- 1: Create a set S of n^h data samples
 - 2: **for all** data samples $s \in S$ **do**
 - 3: Run min-cost filter ordering algorithm for s
 - 4: **if** this run invokes filter j **then**
 - 5: $X_s(\mathcal{T}_\ell, j) = 1$
 - 6: **else**
 - 7: $X_s(\mathcal{T}_\ell, j) = 0$
 - 8: **end if**
 - 9: **end for**
 - 10: Compute $\forall j, \tilde{A}(\mathcal{T}_\ell, j) = \max\left\{\frac{\sum_{s \in S} X_s(\mathcal{T}_\ell, j)}{|S|}, \frac{1}{n^\omega}\right\}$
 - 11: **return** $\forall j, \tilde{A}(\mathcal{T}_\ell, j)$
-

5. ANALYSIS OF VCV

Our goal in this section is to prove that VCV is indeed a (τ, ϕ) -approximation scheme for the flow maximization problem. For ease of analysis, we make no attempts to optimize the various parameters and constants involved in the analysis; in particular, we believe, we can demonstrate a substantially better runtime without any modifications to the algorithm, through a tighter analysis – we defer this task to a full version of this paper. We begin by recalling that the primal LP we are interested in solving is the following:

$$\max \sum_{\mathcal{T} \in \mathcal{D}} f(\mathcal{T}) \quad (5)$$

$$\forall filters\ j : \frac{\sum_{\mathcal{T} \in \mathcal{D}} f(\mathcal{T}) \cdot A(\mathcal{T}, j)}{b(j)} \leq 1 \quad (6)$$

Consider the dual LP:

$$\min \sum_j y(j)b(j) \quad (7)$$

$$\forall \mathcal{T} \in \mathcal{D} : \sum_j y(j) \cdot A(\mathcal{T}, j) \geq 1 \quad (8)$$

The dual LP has a natural interpretation as follows. Variable $y(j)$ denotes the (virtual) *cost* of evaluating filter j . Our goal is to assign costs to filters such that the total weighted cost $\sum_j y(j)b(j)$ is minimized subject to the constraint that the expected cost of each decision tree is at least

one. We emphasize that, as is generally true in primal-dual algorithms, the costs in the dual LP are conceptual and do not necessarily have a physical interpretation.

Given a cost-vector y , recall that in Step 3 of VCV, we define the dual-value $Q(y) \stackrel{\text{def}}{=} \sum_j y(j)b(j)$. Define $\alpha(y)$ as the cost of the minimum cost tree under the cost-vector y : $\alpha(y) \stackrel{\text{def}}{=} \min_{\mathcal{T} \in \mathcal{D}} \sum_j A(\mathcal{T}, j)y(j)$. The dual problem can now be conveniently stated as that of finding a cost-vector y such that $\frac{Q(y)}{\alpha(y)}$ is minimized. Define $\zeta \stackrel{\text{def}}{=} \min_y \frac{Q(y)}{\alpha(y)}$. By the theorem of LP-duality, ζ is the optimal value of our flow problem.

Before proceeding further, we first state the Chernoff-Hoeffding tail bounds which will be useful in our analysis.

FACT 3. ([5, 14]) *Given independent random variables $X_1, \dots, X_u \in [0, 1]$, let $X = \sum_{i=1}^u X_i$ and $\mu = \mathbf{E}[X]$; then (i) for any $\psi > 0$, $\Pr[X \geq (1 + \psi)\mu] < \left(\frac{e^\psi}{(1+\psi)(1+\psi)}\right)^\mu$, (ii) for any $\psi \in [0, 1]$, $\Pr[X \leq (1 - \psi)\mu] \leq \left(\frac{e^{-\psi}}{(1-\psi)(1-\psi)}\right)^\mu$, and (iii) for any $\psi \in [0, 1]$, $\Pr[|X - \mu| \geq \psi\mu] \leq 2e^{-\frac{\mu\psi^2}{3}}$.*

We begin by upper bounding the number of iterations in VCV.

LEMMA 4. *The number of iterations in VCV is at most $\ell_{\max} = n \lfloor \frac{1}{\epsilon} (1 + \log_{1+\epsilon}(n)) \rfloor$.*

PROOF. Suppose the VCV algorithm terminates after t iterations (i.e., it returns t cost vectors). Consider the ℓ^{th} iteration of VCV: we increase the cost of the bottleneck filter exactly by a factor of $1 + \epsilon$, and increase the cost of the other filters by at most this factor. Further, for any filter j , (3) implies $y_1(j) = \frac{\delta}{b(j)}$ and our stopping condition implies $y_{t+1}(j) < \frac{1+\epsilon}{b(j)}$; hence, the number of iterations in which j is the bottleneck filter is at most $\lfloor \log_{1+\epsilon} \left(\frac{1+\epsilon}{\delta}\right) \rfloor$, which by (3) is at most $\lfloor \frac{1}{\epsilon} \cdot (1 + \log_{1+\epsilon}(n)) \rfloor$. Since there are n filters in all, the total number of iterations ℓ_{\max} in the VCV algorithm is at most $\ell_{\max} = n \lfloor \frac{1}{\epsilon} \cdot (1 + \log_{1+\epsilon}(n)) \rfloor$. \square

LEMMA 5. *Consider a fixed invocation of **Sample-and-Estimate** in Step 5 of Algorithm 1 for the value $\hat{A}(\mathcal{T}_\ell, j)$. Let $A(\mathcal{T}_\ell, j) \geq \frac{1}{n^\omega}$. Consider any $\beta \geq 1$; if we set $h = \omega + 3\beta + 3$ in Step 1 of **Sample-and-Estimate**, then $\Pr \left[|\hat{A}(\mathcal{T}_\ell, j) - A(\mathcal{T}_\ell, j)| \geq \frac{A(\mathcal{T}_\ell, j)}{n^\beta} \right] \leq e^{-n^\beta}$.*

PROOF.

$$\begin{aligned}
& \Pr \left[\left| \sum_{s \in S(\mathcal{T}_\ell, j)} X_s(\mathcal{T}_\ell, j) - \mathbf{E} \left[\sum_{s \in S(\mathcal{T}_\ell, j)} X_s(\mathcal{T}_\ell, j) \right] \right| \right. \\
& \geq \frac{\mathbf{E} \left[\sum_{s \in S(\mathcal{T}_\ell, j)} X_s(\mathcal{T}_\ell, j) \right]}{n^\beta} \left. \right] \\
& \leq 2e^{-\frac{\mathbf{E} \left[\sum_{s \in S(\mathcal{T}_\ell, j)} X_s(\mathcal{T}_\ell, j) \right]}{3n^{2\beta}}} \quad (\text{from Fact 3}) \\
& \leq 2e^{-\frac{n^{h-\omega-2\beta}}{3}} \\
& \leq e^{-n^{h-\omega-2\beta-3}} \\
& \quad (\text{since } h = \omega + 3\beta + 3, n \geq 2, \text{ and } \beta \geq 1) \\
& = e^{-n^\beta}
\end{aligned} \tag{9}$$

Further, observe that:

$$\begin{aligned}
& |\tilde{A}(\mathcal{T}_\ell, j) - A(\mathcal{T}_\ell, j)| \geq \frac{A(\mathcal{T}_\ell, j)}{n^\beta} \\
\Rightarrow & \left| \max \left\{ \sum_{s \in S(\mathcal{T}_\ell, j)} \frac{X_s(\mathcal{T}_\ell, j)}{n^h}, \frac{1}{n^\omega} \right\} - A(\mathcal{T}_\ell, j) \right| \geq \frac{A(\mathcal{T}_\ell, j)}{n^\beta} \\
\Rightarrow & \left| \max \left\{ \sum_{s \in S(\mathcal{T}_\ell, j)} X_s(\mathcal{T}_\ell, j), \frac{n^h}{n^\omega} \right\} - n^h \cdot A(\mathcal{T}_\ell, j) \right| \\
& \geq n^h \cdot \frac{A(\mathcal{T}_\ell, j)}{n^\beta} \\
\Rightarrow & \left| \max \left\{ \sum_{s \in S(\mathcal{T}_\ell, j)} X_s(\mathcal{T}_\ell, j), \frac{n^h}{n^\omega} \right\} - \mathbf{E} \left[\sum_{s \in S(\mathcal{T}_\ell, j)} X_s(\mathcal{T}_\ell, j) \right] \right| \\
& \geq \frac{\mathbf{E} \left[\sum_{s \in S(\mathcal{T}_\ell, j)} X_s(\mathcal{T}_\ell, j) \right]}{n^\beta}
\end{aligned} \tag{10}$$

Assuming that the event in the R.H.S. of (10) occurred, one of the below two mutually exclusively events must have occurred: (a) $\sum_{s \in S(\mathcal{T}_\ell, j)} X_s(\mathcal{T}_\ell, j) \geq \frac{n^h}{n^\omega}$; in this case, we have: $\left| \sum_{s \in S(\mathcal{T}_\ell, j)} X_s(\mathcal{T}_\ell, j) - \mathbf{E} \left[\sum_{s \in S(\mathcal{T}_\ell, j)} X_s(\mathcal{T}_\ell, j) \right] \right| \geq \frac{\mathbf{E} \left[\sum_{s \in S(\mathcal{T}_\ell, j)} X_s(\mathcal{T}_\ell, j) \right]}{n^\beta}$, or (b) $\sum_{s \in S(\mathcal{T}_\ell, j)} X_s(\mathcal{T}_\ell, j) < \frac{n^h}{n^\omega}$; in this case, since $A(\mathcal{T}_\ell, j) \geq \frac{1}{n^\omega}$, we have: $\left| \sum_{s \in S(\mathcal{T}_\ell, j)} X_s(\mathcal{T}_\ell, j) - \mathbf{E} \left[\sum_{s \in S(\mathcal{T}_\ell, j)} X_s(\mathcal{T}_\ell, j) \right] \right| \geq \left| \frac{n^h}{n^\omega} - \mathbf{E} \left[\sum_{s \in S(\mathcal{T}_\ell, j)} X_s(\mathcal{T}_\ell, j) \right] \right| \geq \frac{\mathbf{E} \left[\sum_{s \in S(\mathcal{T}_\ell, j)} X_s(\mathcal{T}_\ell, j) \right]}{n^\beta}$. In either case, we have:

$$\begin{aligned}
& |\tilde{A}(\mathcal{T}_\ell, j) - A(\mathcal{T}_\ell, j)| \geq \frac{A(\mathcal{T}_\ell, j)}{n^\beta} \\
\Rightarrow & \left| \sum_{s \in S(\mathcal{T}_\ell, j)} X_s(\mathcal{T}_\ell, j) - \mathbf{E} \left[\sum_{s \in S(\mathcal{T}_\ell, j)} X_s(\mathcal{T}_\ell, j) \right] \right| \\
& \geq \frac{\mathbf{E} \left[\sum_{s \in S(\mathcal{T}_\ell, j)} X_s(\mathcal{T}_\ell, j) \right]}{n^\beta}
\end{aligned}$$

Hence,

$$\begin{aligned}
& \Pr \left[|\tilde{A}(\mathcal{T}_\ell, j) - A(\mathcal{T}_\ell, j)| \geq \frac{A(\mathcal{T}_\ell, j)}{n^\beta} \right] \\
& \Pr \left[\left| \sum_{s \in S(\mathcal{T}_\ell, j)} X_s(\mathcal{T}_\ell, j) - \mathbf{E} \left[\sum_{s \in S(\mathcal{T}_\ell, j)} X_s(\mathcal{T}_\ell, j) \right] \right| \right. \\
& \geq \frac{\mathbf{E} \left[\sum_{s \in S(\mathcal{T}_\ell, j)} X_s(\mathcal{T}_\ell, j) \right]}{n^\beta} \left. \right] \\
& \leq e^{-n^\beta} \quad \{\text{from (9)}\}
\end{aligned}$$

This completes the proof of the lemma. \square

LEMMA 6. *Consider a fixed invocation of **Sample-and-Estimate** in Step 5 of Algorithm 1 for the value $\hat{A}(\mathcal{T}_\ell, j)$. Let $A(\mathcal{T}_\ell, j) \leq \frac{1}{n^\omega}$. Consider any $\beta \geq 1$; if we set $h = \omega + 3\beta + 3$ in Step 1 of **Sample-and-Estimate**, then $\Pr \left[\hat{A}(\mathcal{T}_\ell, j) \geq (1 + \frac{1}{n^\beta}) \cdot \frac{1}{n^\omega} \right] \leq e^{-n^\beta}$.*

PROOF. Clearly, $\Pr \left[\hat{A}(\mathcal{T}_\ell, j) \geq (1 + \frac{1}{n^\beta}) \cdot \frac{1}{n^\omega} \right]$ is an increasing function of $A(\mathcal{T}_\ell, j)$, and since we assumed $A(\mathcal{T}_\ell, j) \leq \frac{1}{n^\omega}$, this probability is maximized when $A(\mathcal{T}_\ell, j) = \frac{1}{n^\omega}$. This fact combined with Lemma 5 completes the proof of this lemma. \square

From Step 10 of **Sample-and-Estimate**, it follows that any estimate $\hat{A}(\mathcal{T}_\ell, j)$ returned by this subroutine is $\geq \frac{1}{n^\omega}$. This fact combined with Lemma 6 yields the following corollary.

COROLLARY 7. Consider a fixed invocation of **Sample-and-Estimate** in Step 5 of Algorithm 1 for the value $\tilde{A}(\mathcal{T}_\ell, j)$. Let $A(\mathcal{T}_\ell, j) \leq \frac{1}{n^\omega}$. Consider any $\beta \geq 1$; if we set $h = \omega + 3\beta + 3$ in Step 1 of **Sample-and-Estimate**, then $\Pr\left[\tilde{A}(\mathcal{T}_\ell, j) \notin \left[\frac{1}{n^\omega}, \left(1 + \frac{1}{n^\beta}\right) \cdot \frac{1}{n^\omega}\right]\right] \leq e^{-n^\beta}$.

Consider a fixed invocation of **Sample-and-Estimate** in Step 5 of Algorithm 1 for the value $\tilde{A}(\mathcal{T}_\ell, j)$. We say that this invocation yields a good estimate if one of the following two conditions hold:

- (i) $A(\mathcal{T}_\ell, j) > \frac{1}{n^\omega}$, and **Sample-and-Estimate** returned an estimate $\tilde{A}(\mathcal{T}_\ell, j)$ such that $|\tilde{A}(\mathcal{T}_\ell, j) - A(\mathcal{T}_\ell, j)| \leq \frac{A(\mathcal{T}_\ell, j)}{n^\beta}$, or
- (ii) $A(\mathcal{T}_\ell, j) \leq \frac{1}{n^\omega}$, and **Sample-and-Estimate** returned an estimate $\tilde{A}(\mathcal{T}_\ell, j)$ such that $\tilde{A}(\mathcal{T}_\ell, j) \in \left[\frac{1}{n^\omega}, \left(1 + \frac{1}{n^\beta}\right) \cdot \frac{1}{n^\omega}\right]$.

LEMMA 8. Consider a fixed invocation of **Sample-and-Estimate** for the value $\tilde{A}(\mathcal{T}_\ell, j)$. If this is a good estimate, then for any $\beta \geq 1$, $\frac{A(\mathcal{T}_\ell, j)}{\tilde{A}(\mathcal{T}_\ell, j)} \leq 1 + \frac{2}{n^\beta} = \rho_1$.

PROOF. From the definition of a good estimate, it follows that if $A(\mathcal{T}_\ell, j) \geq \frac{1}{n^\omega}$, then $\frac{A(\mathcal{T}_\ell, j)}{\tilde{A}(\mathcal{T}_\ell, j)} \leq \frac{1}{1 - \frac{1}{n^\beta}}$; else if $A(\mathcal{T}_\ell, j) < \frac{1}{n^\omega}$, then $\frac{A(\mathcal{T}_\ell, j)}{\tilde{A}(\mathcal{T}_\ell, j)} \leq 1$. In either case, we have $\frac{A(\mathcal{T}_\ell, j)}{\tilde{A}(\mathcal{T}_\ell, j)} \leq \frac{1}{1 - \frac{1}{n^\beta}}$; it is verify that this is upper bounded by $1 + \frac{2}{n^\beta}$, for any $n \geq 2$ and $\beta \geq 1$. This completes the proof of the lemma. \square

LEMMA 9. Suppose we set $\beta = \frac{1}{\log 2} \cdot \max\left\{\log\left(2 \cdot \log\left(\frac{2}{\epsilon \log(1+\epsilon)}\right)\right), \frac{\log(6-2 \cdot \frac{\log(1-\phi)}{\log 2})}{\log 2} + 1\right\}$, and set $h = \omega + 3\beta + 3$ in Step 1 of **Sample-and-Estimate**. Consider the event that “every invocation of **Sample-and-Estimate** by Algorithm 1 returned a good estimate”. The probability of this event occurring is at least ϕ .

PROOF. By Lemma 4, there are at most $n \lfloor \frac{1}{\epsilon} (1 + \log_{1+\epsilon}(n)) \rfloor$ iterations in Algorithm 1; each iteration invokes **Sample-and-Estimate** at most n times. Hence, there are at most $n^2 \lceil \frac{1}{\epsilon} (1 + \log_{1+\epsilon}(n)) \rceil$ invocations of **Sample-and-Estimate** before the termination of Algorithm 1. This fact, Lemmas 5 and 6, and the union bound together imply that the probability of the event under consideration *not* occurring is at most $n^2 \lceil \frac{1}{\epsilon} (1 + \log_{1+\epsilon}(n)) \rceil \cdot e^{-n^\beta}$. Define $\rho = -\frac{\log(1-\phi)}{\log 2}$.

We have:

$$\begin{aligned} \beta &= \\ & \frac{1}{\log 2} \cdot \max\left\{\log\left(2 \cdot \log\left(\frac{2}{\epsilon \log(1+\epsilon)}\right)\right), \frac{\log(6+2\rho)}{\log 2} + 1\right\} \\ & \Rightarrow \\ & \beta \geq \\ & \frac{1}{\log 2} \cdot \max\left\{\log \log\left(\frac{2}{\epsilon \log(1+\epsilon)}\right)^2, \frac{\log \log(n^{3+\rho})^2}{\log n}\right\} \\ & \Rightarrow \\ & \beta \geq \frac{\log \log \max\left\{\frac{2}{\epsilon \log(1+\epsilon)}, n^{2+\rho} \cdot \log n\right\}^2}{\log n} \\ & \quad (\text{since } n \geq 2 \text{ and } n \geq \log(n)) \\ & \Rightarrow \\ & \beta \geq \frac{\log \log\left(\frac{2}{\epsilon} \cdot \log_{1+\epsilon}(n) \cdot n^{2+\rho}\right)}{\log n} \\ & \Rightarrow \\ & n^\beta \geq \log\left(\frac{2}{\epsilon} \cdot \log_{1+\epsilon}(n) \cdot n^{2+\rho}\right) \\ & \Rightarrow \\ & \log \frac{2}{\epsilon} + \log \log_{1+\epsilon}(n) - n^\beta \leq -(2+\rho) \log n \\ & \Rightarrow \\ & \left(\frac{2}{\epsilon} \cdot \log_{1+\epsilon}(n)\right) \cdot e^{-n^\beta} \leq n^{-2-\rho} \\ & \Rightarrow \\ & \frac{n^2}{\epsilon} \cdot (1 + \log_{1+\epsilon}(n)) \cdot e^{-n^\beta} \leq 1 - \phi \\ & \Rightarrow \\ & n^2 \cdot \left\lfloor \frac{1}{\epsilon} (1 + \log_{1+\epsilon}(n)) \right\rfloor \cdot e^{-n^\beta} \leq 1 - \phi \end{aligned}$$

This completes the proof of the Lemma. \square

We now show that the flow output by our algorithm is feasible.

LEMMA 10. If every invocation of **Sample-and-Estimate** by VCV returned a good estimate, then VCV algorithm outputs a feasible flow.

PROOF. Assume that VCV outputs a flow which consists of t cost-vectors, where $t \leq \ell_{\max}$ is a fixed positive integer. Recall that the flow value corresponding to cost-vector y_ℓ is $f(\mathcal{T}_\ell) = \frac{r(\mathcal{T}_\ell)}{\rho_1 \cdot \log_{1+\epsilon-\epsilon^2} \frac{1+\epsilon}{\delta}}$, where $r(\mathcal{T}_\ell)$ is the bottleneck flow value for cost-vector $y(\ell)$. Let us *pretend* that we did *not* scale down the flows in Step 12 of the VCV algorithm: i.e., assume we set $f(\mathcal{T}_\ell) = r(\mathcal{T}_\ell)$ in Step 12 of the VCV algorithm. Under this setting, we will show that the L.H.S. of the j^{th} constraint in (6) in the primal LP is at most $\rho_1 \cdot \log_{1+\epsilon-\epsilon^2} \frac{1+\epsilon}{\delta}$. In reality, since we do scale our final flows by precisely this factor, the final flow output by our algorithm is indeed feasible.

Consider a specific iteration $\ell \leq t$ of VCV. In this iteration, we select a cost-vector y_ℓ and increase the total (unscaled) flow by $r(\mathcal{T}_\ell) = \frac{b(\mathcal{T}_\ell^{j^{\min}})}{\tilde{A}_{\mathcal{T}_\ell}(j^{\min})}$. If $t < \ell \leq \ell_{\max}$, then without loss of generality, we will assume that $y(\ell) = y(t)$ and $r(\mathcal{T}_\ell) = 0$. Simultaneously, for filter j in the primal LP, we increase the L.H.S. of the j^{th} row by a value of at most $\rho_1 \cdot z_\ell(j)$, where $z_\ell(j) = \frac{\tilde{A}_{\mathcal{T}_\ell}(j) r(\mathcal{T}_\ell)}{b(j)}$ (by Lemma 8). We also increase the dual cost $y(j)$ by a factor of $1 + \epsilon z_\ell(j)$. By our choice of j_ℓ^{\min} , for all ℓ and j , we have $z_\ell(j) \leq 1$. By

our stopping condition, $Q_t = \sum_j y_t(j)b(j) < 1 \leq Q_{t+1} = \sum_j y_{\mathcal{T}_{t+1}}(j)b(j)$. This inequality, combined with the fact that the cost of any filter increases by a factor of at most $1 + \epsilon$ in successive iteration yields: $\forall j : y_t(j) < \frac{1}{b(j)}$ and $y_{t+1}(j) < \frac{1+\epsilon}{b(j)}$. Since $y_1(j) = \frac{\delta}{b(j)}$, we have:

$$\begin{aligned}
& \frac{\delta}{b(j)} \cdot \prod_{\ell=1}^{\ell_{\max}} (1 + \epsilon z_\ell(j)) < \frac{1 + \epsilon}{b(j)} \\
& \Rightarrow \prod_{\ell=1}^{\ell_{\max}} (1 + \epsilon z_\ell(j)) < \frac{1 + \epsilon}{\delta} \\
& \Rightarrow \prod_{\ell=1}^{\ell_{\max}} e^{\epsilon z_\ell(j) - (\epsilon z_\ell(j))^2} < \frac{1 + \epsilon}{\delta} \\
& \{\text{Since } \epsilon z_\ell(j) \leq \frac{1}{2}, \text{ and } \forall x \in [0, \frac{1}{2}], e^{x-x^2} \leq 1 + x\} \\
& \Rightarrow \prod_{\ell=1}^{\ell_{\max}} e^{(\epsilon - \epsilon^2) \cdot z_\ell(j)} < \frac{1 + \epsilon}{\delta} \{\text{since } z_\ell(j) \leq 1\} \\
& \Rightarrow e^{(\epsilon - \epsilon^2) \cdot \sum_{\ell=1}^{\ell_{\max}} z_\ell(j)} < \frac{1 + \epsilon}{\delta} \\
& \Rightarrow (1 + \epsilon - \epsilon^2)^{\sum_{\ell=1}^{\ell_{\max}} z_\ell(j)} < \frac{1 + \epsilon}{\delta} \\
& \{\text{Since } \forall x, 1 + x \leq e^x\} \\
& \Rightarrow \sum_{\ell=1}^{\ell_{\max}} z_\ell(j) < \log_{1+\epsilon-\epsilon^2} \frac{1 + \epsilon}{\delta} \\
& \Rightarrow \rho_1 \cdot \sum_{\ell=1}^{\ell_{\max}} z_\ell(j) < \rho_1 \cdot \log_{1+\epsilon-\epsilon^2} \frac{1 + \epsilon}{\delta}
\end{aligned}$$

Since, we scale down the flow values precisely by the factor $\rho_1 \cdot \log_{1+\epsilon-\epsilon^2} \frac{1+\epsilon}{\delta}$, the final flow output by VCV is indeed feasible. \square

LEMMA 11. Suppose we set $\omega = \beta + \lambda_0 + \frac{2}{\epsilon} + 1$ in Step 10 of *Sample-and-Estimate*. Consider a fixed iteration ℓ of VCV and the corresponding cost-vector y_ℓ that is returned by VCV. Assuming “every invocation of *Sample-and-Estimate* returned a good estimate”, y_ℓ satisfies the following inequality: $\sum_j y_\ell(j) \tilde{A}(\mathcal{T}_\ell, j) \leq \rho_2 \gamma \alpha(y_\ell)$, where $\rho_2 = 1 + \frac{3}{n^\beta}$.

PROOF. Define $z_\ell = \sum_j y_\ell(j) A(\mathcal{T}_\ell, j)$. We first note that, from our assumption about the γ -approximation of the min-cost filter ordering algorithm, it follows that:

$$z_\ell \leq \gamma \alpha(y_\ell) \quad (11)$$

Second, since at least one filter needs to be evaluated by the min-cost filter ordering algorithm, it follows that $z_\ell \geq \min_j y_\ell(j) \geq \min_j y_1(j) \geq n^{-\lambda_0 - \frac{2}{\epsilon}}$; the last inequality follows from (3). By our stopping condition it also follows $\max_j y_\ell(j) \leq \frac{1}{\min_j b(j)} = 1$.

These facts imply:

$$\begin{aligned}
& \sum_j y_\ell(j) \tilde{A}(\mathcal{T}_\ell, j) = \\
& \sum_{j: A(\mathcal{T}_\ell, j) \leq \frac{1}{n^\omega}} y_\ell(j) \tilde{A}(\mathcal{T}_\ell, j) \\
& + \sum_{j: A(\mathcal{T}_\ell, j) > \frac{1}{n^\omega}} y_\ell(j) A(\mathcal{T}_\ell, j) \\
& \leq \sum_{j: A(\mathcal{T}_\ell, j) \leq \frac{1}{n^\omega}} y_\ell(j) \cdot (1 + \frac{1}{n^\beta}) \cdot \frac{1}{n^\omega} \\
& + \sum_{j: A(\mathcal{T}_\ell, j) > \frac{1}{n^\omega}} y_\ell(j) \cdot A(\mathcal{T}_\ell, j) \cdot (1 + \frac{1}{n^\beta}) \\
& \{\text{from defn. of good estimates}\} \\
& \leq n \cdot (1 + \frac{1}{n^\beta}) \cdot \frac{1}{n^\omega} \\
& + \sum_{j: A(\mathcal{T}_\ell, j) > \frac{1}{n^\omega}} y_\ell(j) \cdot A(\mathcal{T}_\ell, j) \cdot (1 + \frac{1}{n^\beta}) \\
& \leq n^{\lambda_0 + \frac{2}{\epsilon} + 1} \cdot z_\ell \cdot (1 + \frac{1}{n^\beta}) \cdot \frac{1}{n^\omega} + z_\ell \cdot (1 + \frac{1}{n^\beta}) \\
& = z_\ell \cdot (1 + \frac{1}{n^\beta}) \cdot (1 + \frac{1}{n^{\omega - \lambda_0 - \frac{2}{\epsilon} - 1}}) \\
& \leq z_\ell \cdot (1 + \frac{3}{n^{\min\{\beta, \omega - \lambda_0 - \frac{2}{\epsilon} - 1\}}}) \\
& = z_\ell \cdot (1 + \frac{3}{n^{\min(\beta, \omega - \lambda_0 - \frac{2}{\epsilon} - 1)}}) \\
& = z_\ell \cdot (1 + \frac{3}{n^\beta})
\end{aligned}$$

Combining the last inequality with (11) completes the proof of the lemma. \square

We now show that our flow value is close to optimal.

LEMMA 12. If every invocation of *Sample-and-Estimate* by VCV returned a good estimate, then our final flow value is within a factor of at most $\frac{\gamma \rho_1 \rho_2}{(1-2\epsilon)^2}$ of the optimal flow value.

PROOF. Consider the ℓ^{th} cost-vector, for any $\ell \geq 1$. We have,

$$\begin{aligned}
Q_{\ell+1} &= \sum_j b(j) y_{\ell+1}(j) \\
&= \sum_j b(j) \cdot y_\ell(j) \cdot \left(1 + \epsilon \cdot \frac{b(j_\ell^{\min}) \tilde{A}_\ell(j)}{b(j) \tilde{A}_\ell(j_\ell^{\min})}\right) \\
&= Q_\ell + \epsilon r_\ell \cdot \sum_j y_\ell(j) \tilde{A}_\ell(j) \\
&\leq Q_\ell + \epsilon r_\ell \alpha(y_\ell) \gamma \rho_2 \quad (\text{by Lemma 11}) \quad (12)
\end{aligned}$$

Eqn. (12) along with induction on ℓ immediately yields:

$$Q_\ell \leq Q_1 + \epsilon \cdot \sum_{j=1}^{\ell-1} r_j \alpha(y_j) \gamma \rho_2 \quad (13)$$

By definition of ζ , we have $\forall j \leq t : \zeta \leq \frac{Q_j}{\alpha(j)}$. Combining this with the fact that $Q_1 = n\delta$, and (13), we get:

$$Q_\ell \leq n\delta + \frac{\epsilon}{\zeta} \cdot \sum_{j=1}^{\ell-1} r_j Q_j \gamma \rho_2 \quad (14)$$

Through induction on ℓ , (14) implies $Q_\ell \leq n\delta \cdot e^{\frac{\epsilon\gamma\rho_2 \cdot \sum_{j=1}^{\ell-1} r_j}{\zeta}}$.
By our stopping condition, $1 \leq Q_{t+1} \leq n\delta \cdot e^{\frac{\epsilon\gamma\rho_2 \sum_{j=1}^t r_j}{\zeta}}$ and hence,

$$\frac{\zeta}{\sum_{j=1}^t r_j} \leq \frac{\epsilon\gamma\rho_2}{\ln(\frac{1}{n\delta})} \quad (15)$$

Thus, the ratio between the optimal flow and our flow is

$$\begin{aligned} \frac{\zeta}{\sum_{j=1}^t f(\mathcal{T}_j)} &= \frac{\zeta\rho_1 \log_{1+\epsilon-\epsilon^2}(\frac{1+\epsilon}{\delta})}{\sum_{j=1}^t r(\mathcal{T}_j)} \\ &\leq \frac{\epsilon\gamma\rho_2}{\ln(\frac{1}{n\delta})} \rho_1 \log_{1+\epsilon-\epsilon^2}\left(\frac{1+\epsilon}{\delta}\right) \{ \text{from (15)} \} \\ &\leq \frac{\epsilon\gamma\rho_1\rho_2 \cdot \frac{1}{\epsilon} \cdot \log((1+\epsilon) \cdot n)}{\log((1+\epsilon) \cdot n)^{\frac{1}{\epsilon}-1} \cdot \log(1+\epsilon-\epsilon^2)} \\ &\quad \{ \text{By definition of } \delta \} \\ &= \frac{\gamma\rho_1\rho_2}{(\frac{1}{\epsilon}-1) \cdot \log(1+\epsilon-\epsilon^2)} \\ &\leq \frac{\gamma\rho_1\rho_2}{(1-\epsilon) \cdot (1-2\epsilon+2\epsilon^2-\epsilon^4)} \\ &\quad \{ \text{Since } \forall x \in [0, \frac{1}{2}], \log(1+x) \geq x-x^2 \} \\ &\leq \frac{\gamma\rho_1\rho_2}{(1-2\epsilon)^2} \end{aligned}$$

□

THEOREM 13. *There exists a value ϵ_0 which is dependent only on τ such that if we set $\epsilon = \epsilon_0$ in the VCV algorithm, then with probability ϕ , we obtain a feasible solution that is within a factor of $\gamma \cdot (1+\tau)$ from the optimal solution. Further, the runtime of VCV is $O(n^\varphi)$, where φ is a fixed constant whose value depends only on τ and ϕ .*

PROOF. From the definition of β , it follows that as $\epsilon \rightsquigarrow 0$, $\beta \rightsquigarrow \infty$, and $\rho_1, \rho_2 \rightsquigarrow 1$. Hence, there exists a positive value ϵ_0 which is dependent only on τ such that if $\epsilon = \epsilon_0$, then $\frac{\rho_1\rho_2}{(1-2\epsilon)^2} \leq (1+\tau)$. Combining this fact along with Lemma 12 yields the first part of the theorem.

The number of iterations in the VCV algorithm is $O(\frac{n \log n}{\epsilon \log(1+\epsilon)})$. The runtime for iteration ℓ is dominated by the time to estimate at most n estimates $\tilde{A}(\mathcal{T}_\ell, j)$; the time taken to estimate $\tilde{A}(\mathcal{T}_\ell, j)$ is the product of the number of samples n^h and the time taken for each sample (which is $O(n)$). Hence, the runtime of VCV is $O(\frac{n \log n}{\epsilon \log(1+\epsilon)} \cdot n \cdot n^h \cdot n)$. The number of samples h is dependent only on β and ϵ , and these two in turn depend only on τ and ϕ . Hence, the theorem follows. □

6. ARBITRARY FILTER PLACEMENTS

The VCV algorithm can be easily extended to the case of arbitrary filter placements, where not only do we need to decide which filters to evaluate, but also which processor to choose for their evaluations. The main modification here is to have the dual-costs be a function of the filter-processor pairs rather than just filters. The cost-update rule is identical as in the case of dedicated filter placements. Further, the scheduling component for the flow maximization problem becomes significantly simplified, as scheduling decisions in the min-cost filter ordering problem can be made trivially: whenever, the min-cost filter ordering algorithm chooses to

evaluate a filter, it does so only on the processor on which the filter incurs the least cost under the given cost-vector. With these modifications, the overall runtime of VCV becomes polynomial in both the number of filters and processors (as opposed to only the number of filters in the case of dedicated filter placement).

7. REFERENCES

- [1] Ron Avnur and Joseph M. Hellerstein. Eddies: continuously adaptive query processing. *SIGMOD Rec.*, 29(2):261–272, 2000.
- [2] Shivnath Babu, Rajeev Motwani, Kamesh Munagala, Itaru Nishizawa, and Jennifer Widom. Adaptive ordering of pipelined stream filters. In *SIGMOD*, pages 407–418, New York, NY, USA, 2004. ACM Press.
- [3] Amotz Bar-Noy, Mihir Bellare, Magnt’us M. Halldt’orsson, Hadas Shachnai, and Tami Tamir. On chromatic sums and distributed resource allocation. *Inf. Comput.*, 140(2):183–202, 1998.
- [4] Surajit Chaudhuri, Umeshwar Dayal, and Tak W. Yan. Join queries with external text sources: execution and optimization techniques. In *SIGMOD ’95: Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, pages 410–422, New York, NY, USA, 1995. ACM.
- [5] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, 23:493–509, 1952.
- [6] Edith Cohen, Amos Fiat, and Haim Kaplan. Efficient sequences of trials. In *SODA*, pages 737–746, 2003.
- [7] Anne Condon, Amol Deshpande, Lisa Hellerstein, and Ning Wu. Flow algorithms for two pipelined filter ordering problems. In *PODS*, pages 193–202, New York, NY, USA, 2006. ACM Press.
- [8] Amol Deshpande, Carlos Guestrin, Wei Hong, and Samuel Madden. Exploiting correlated attributes in acquisitional query processing. In *ICDE ’05: Proceedings of the 21st International Conference on Data Engineering*, pages 143–154, Washington, DC, USA, 2005. IEEE Computer Society.
- [9] Oren Etzioni, Steve Hanks, Tao Jiang, Richard M. Karp, Omid Madani, and Orli Waarts. Efficient information gathering on the internet (extended abstract). In *FOCS*, pages 234–243, 1996.
- [10] Uriel Feige and Prasad Tetali. Approximating min sum set cover. *Algorithmica*, 40(4):219–234, 2004.
- [11] N. Garg and J. Koenemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, page 300. IEEE Computer Science Society, 1998.
- [12] Roy Goldman and Jennifer Widom. Wsq/dsq: a practical approach for combined querying of databases and the web. In *SIGMOD ’00: Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 285–296, New York, NY, USA, 2000. ACM.
- [13] J. Hellerstein and M. Stonebraker. Predicate migration: Optimizing queries with expensive predicates. In *Proc. SIGMOD*, 1993.

- [14] W. Hoeffding. Probability inequalities for sums of bounded random variables. *American Statistical Association Journal*, 58:13–30, 1963.
- [15] T. Ibaraki and T. Kameda. On the optimal nesting order for computing n-relational joins. *ACM Trans. on Database Systems*, 9(3):482–502, 1984.
- [16] Haim Kaplan, Eyal Kushilevitz, and Yishay Mansour. Learning with attribute costs. In *STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 356–365, New York, NY, USA, 2005. ACM.
- [17] Murali S. Kodialam. The throughput of sequential testing. In *Proceedings of the 8th International IPCO Conference on Integer Programming and Combinatorial Optimization*, pages 280–292, London, UK, 2001. Springer-Verlag.
- [18] Ravi Krishnamurthy, Haran Boral, and Carlo Zaniolo. Optimization of nonrecursive queries. In *Proc. VLDB*, pages 128–137, 1986.
- [19] Zhen Liu, Srinivasan Parthasarathy, Anand Ranganathan, and Hao Yang. Near-optimal algorithms for shared filter evaluation in data stream systems. In *Proc. of ACM SIGMOD (to appear)*, 2008.
- [20] Kamesh Munagala, Shivnath Babu, Rajeev Motwani, and Jennifer Widom. The pipelined set cover problem. In *10th International Conference on Database Theory - ICDT*, pages 83–98, 2005.
- [21] Kamesh Munagala, Utkarsh Srivastava, and Jennifer Widom. Optimization of continuous queries with shared expensive filters. In *PODS '07: Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 215–224, New York, NY, USA, 2007. ACM.
- [22] Serge A. Plotkin, David B. Shmoys, and Éva Tardos. Fast approximation algorithms for fractional packing and covering problems. In *Proceedings of the 32nd annual symposium on Foundations of computer science*, pages 495–504, Los Alamitos, CA, USA, 1991. IEEE Computer Society Press.
- [23] H. Simon and J. Kadane. Optimal problem-solving search: All-or-none solutions. *Artificial Intelligence*, 6:235–247, 1975.
- [24] Vijay V. Vazirani. *Approximation algorithms*. Springer-Verlag New York, Inc., New York, NY, USA, 2001.
- [25] Neal E. Young. Sequential and parallel algorithms for mixed packing and covering. In *Proceedings of IEEE Symposium on Foundations of Computer Science*, 2001.