Flow Algorithms for Two Pipelined Filter Ordering Problems*

Anne Condon
Dept. of Computer Science
U. British Columbia
condon@cs.ubc.ca

Amol Deshpande
Dept. of Computer Science
U. Maryland
amol@cs.umd.edu

Lisa Hellerstein, Ning Wu
Dept. of Computer and Information Science
Polytechnic University
hstein,wning@cis.poly.edu

ABSTRACT

Pipelined filter ordering is a central problem in database query optimization, and has received renewed attention recently in the context of environments such as the web, continuous high-speed data streams and sensor networks. We present algorithms for two natural extensions of the classical pipelined filter ordering problem: (1) a *distributional type* problem where the filters run in parallel and the goal is to maximize throughput, and (2) an *adversarial type* problem where the goal is to minimize the expected value of *multiplicative regret*. We show that both problems can be solved using similar flow algorithms, which find an optimal ordering scheme in time $O(n^2)$, where n is the number of filters. Our algorithm for (1) improves on an earlier $O(n^3 \log n)$ algorithm of Kodialam.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query processing*; F.2.0 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*General*

General Terms

Algorithms

Keywords

pipelined filter ordering, selection ordering, query optimization, flow algorithms

*Research conducted in part while L. Hellerstein was visiting Univ. of Wisc., Madison, and Univ. of British Columbia. L. Hellerstein and N. Wu were partially supported by NSF grant ITR-0205647. Hellerstein also received support from the Othmer Institute for Interdisciplinary Studies. A. Condon was supported by an NSERC grant and by the Peter Wall Institute for Advanced Studies at UBC.

PODS'06, June 26–28, 2006, Chicago, Illinois, USA.

1. INTRODUCTION

Pipelined filter ordering (sometimes called *selection ordering*) refers to the problem of determining the order in which to apply a given set of commutative filters (predicates) to all the tuples of a relation, so as to find the tuples that satisfy all the filters. In addition to conjunctive selection queries, many commonly occurring join queries (specifically, those posed against a *star* schema) reduce to pipelined filter ordering. In recent years, pipelined filter ordering problems have received renewed attention in the context of environments such as the web [5, 14, 9], continuous high-speed data streams [1, 3], and sensor networks [8]. These environments present significantly different challenges, and cost structures, than do traditional centralized database systems. Pipelined filter ordering problems are also studied in other areas such as fault detection and machine learning (see e.g. Shayman et al. [20] and Kaplan et al. [15]), under names such as learning with attribute costs [15], minimum-sum set cover [10], and satisficing search [21]. **In this paper, we present efficient algorithms for two pipelined filter ordering problems, both originally motivated by questions of database query optimization.** One problem is new; the other was already studied in a different context by Kodialam [17], and we obtain improved running time.

Pipelined filter ordering problems can be partitioned into two types. In the "distributional" type, assumptions are made about the probability that tuples will satisfy a given filter, and optimization is with respect to expected behavior. Probabilities may be learned from the statistics maintained on the tables, or on-the-fly while tuples are being processed [1, 3]. In the "adversarial" type, the goal is to optimize with respect to worst-case assumptions, such as when an adversary controls which tuples satisfy which filters.

The classical pipelined filter ordering problem [18] is a distributional type of problem. A cost and probability are given for each filter – the cost of applying the filter to a tuple, and the probability that the tuple satisfies the filter and is not eliminated. The event that a tuple satisfies a filter is independent of whether the tuple satisfies other filters. The problem is to find the ordering ("pipeline") of filters that yields minimum expected cost for processing a tuple. A simple polynomial-time solution is to order the filters in non-decreasing order of the ratio $c_i/(1-p_i)$, where p_i is the probability associated with filter i and c_i is the cost of applying filter i (cf. [12, 21, 18]). Variants of the classical

problem, such as finding the single best ordering for a given set of tuples, have been studied recently, We discuss these in more detail in Section 2.

In this paper we present algorithms for two natural extensions of pipelined filter ordering problems. We introduce the problems here and provide formal details in Sections 3 and 4. Since our interest in filter ordering was motivated by the problem of ordering database selection queries, we discuss our work in this context, although it can be interpreted more generally.

• **Distributional type, parallel environment:** Our first result is for a distributional type problem in a parallel or distributed environment. Our interest in this problem is motivated by two increasingly prevalent scenarios: (1) massively parallel database systems and (2) web-based structured information sources such as IMDB and Amazon. In both, selection queries (i.e. conjunctions of predicates, or filters) may be processed in parallel as follows. For each predicate of the query, there is a distinct operator (processor) dedicated to evaluating that predicate. Each tuple in the input relation is routed from operator to operator, until it is found to satisfy all predicates of the query and is output, or until it is found not to satisfy a predicate, in which case it is discarded. Each tuple can be routed individually, so that different tuples can have different routes. At any moment, each operator can evaluate its predicate on at most one tuple, and each tuple can be evaluated by at most one processor; but the n different operators can work in parallel on n different tuples. The problem, then, is to determine how best to route each tuple. In solving this problem, we assume that the selectivity of each operator O_i, i.e. the probability p_i that a tuple satisfies O_i's predicate, is known, and that each operator O_i has a known rate limit r_i on the expected number of tuples it can process per unit time. (This formulation is equivalent to one in which r_i is defined to be the maximum, rather than the expected, number of tuples that O_i can process in unit time, and excess tuples are queued; see the discussion of Kodialam's results in Section 2.) We also assume that the event that a tuple satisfies a predicate is independent of whether the tuple satisfies any of the other predicates, and of the events that other tuples satisfy any predicates. Our goal is to route tuples so as to maximize the throughput of tuple handling, subject to the constraint that the expected number of tuples processed by each operator O_i per unit time does not exceed r_i. We call this the *max-throughput* problem.

Kodialam [17] gave algorithms that, given an instance of the max-throughput problem (i.e. the selectivities and rate limits of each of the n operators), find (1) the max throughput, and (2) an optimal routing scheme. His algorithms run in time $O(n^2)$ and $O(n^3 \log n)$ respectively; they exploit the polymatroid structure of a certain space associated with the problem instance, and build on a constructive proof of the Caratheodory representation theorem. We present an algorithm for (1) that runs in linear time if the rate limits $\{r_i\}$ are given in sorted order and an algorithm for (2) that runs in $O(n^2)$ time. Our algorithms are conceptually simpler than Kodialam's; we use a flow-based algorithm for (2) and our analysis of this algorithm provides the basis for our linear-time algorithm for (1).

• **Adversarial type, single tuple:** Our second result pertains to a new, adversarial type of problem. We focus on the problem of routing a single tuple through the operators, where a cost c_i is associated with each operator O_i. If a tuple is processed by operators $O_{i_1}, O_{i_2}, \ldots, O_{i_k}$ before being eliminated by O_{i_k}, then the total cost of processing the tuple is $c_{i_1} + \ldots + c_{i_k}$. Had the tuple been routed to O_{i_k} first, it would have incurred a cost of only c_{i_k}. The *multiplicative regret* is $\frac{c_{i_1}+\ldots+c_{i_k}}{c_{i_k}}$, the ratio of the actual cost incurred in processing the tuple, to the minimum possible cost that could be incurred under an optimal routing of that tuple.

The problem is to choose a (randomized) routing of the tuple so as to minimize the expected multiplicative regret, under the following assumptions. We assume that the set of filters which will eliminate the tuple are determined (in secret) by an adversary before a routing is chosen for the tuple. The goal of the adversary is to maximize the expected multiplicative regret induced by the tuple routing. The adversary (who may make random choices) will know the strategy used in determining the randomized routing of the tuple, and can choose the set of filters accordingly. We thus have a classical zero-sum game between two players – the routing player and the adversary – and the problem is to determine the optimal strategy of the routing player. We call this the *game theoretic multiplicative regret* (GTMR) problem. Our algorithm for the GTMR problem is based on the same flow techniques that we use for the max-throughput problem and runs in time $O(n^2)$.

In what follows, we actually use an equivalent formulation of the GTMR problem, in which we restrict the adversary to choose exactly one filter to eliminate the tuple. The equivalence follows from that fact that the restriction does not disadvantage the adversary. It is easy to show that it is not in the interest of the adversary to cause the tuple to satisfy all filters (because then the multiplicative regret is 1, which is the minimum possible), nor to choose more than one filter to eliminate a tuple (because if S is the set of filters that eliminate the tuple, removing all but the lowest cost filter in S can only increase multiplicative regret).

From a practical point of view, assumption of such a powerful adversary is not well motivated, since real-world data tends not to have worst-case properties. However, from a theoretical perspective, our analysis provides insight into worst-case behavior of pipelined filtering with costs. We note that the assumption of such an adversary is standard in on-line optimization problems, in which the goal is to minimize the competitive ratio (which is a type of multiplicative regret). The GTMR problem is not a proper on-line problem, however, since it takes only a single input, rather than a sequence of inputs.

A naive strategy for minimizing multiplicative regret routes the tuples through the operators in increasing order of their costs. As noted by Kaplan et al. [15], this strategy incurs a multiplicative regret of at most n. How much worse is this strategy than the optimal strategy returned by our GTMR algorithm? If all costs are equal, then the adversary will cause the optimal strategy to have (expected) multiplicative regret $(n+1)/2$, and the naive strategy to have multiplicative regret of n. We show that, for any set of costs, the naive strategy achieves multiplicative regret that is within

a factor 2 of the expected multiplicative regret achieved by the optimal strategy.

We can also show (details omitted) that variants of the GTMR problem, in which the goal is to minimize additive regret or total cost, rather than multiplicative regret, have simple linear-time algorithms, assuming sorted input.

Following a discussion of related work, we present our main results on the the max-throughput problem and the game theoretic multiplicative regret problem in Sections 3 and 4 respectively.

2. RELATED WORK

Kodialam [17] gave an algorithm for the max-throughput problem, but with higher running time than the algorithm given in this paper. He first introduces a problem variant that takes queueing delays into account. Note that our formulation of the max-throughput problem implicitly assumes that an operator O_i can sometimes process tuples at a rate that exceeds its limit r_i, since a solution only guarantees that the *expected* rate of tuples arriving at O_i will not exceed r_i. Kodialam's queueing-theory formulation imposes a limit on *maximum*, rather than expected, rates, with excess tuples at operators buffered in queues. Following early work of Coffman and Mitrani [6] and Gelenbe and Mitrani [13], Kodialam reduces the queueing-theory formulation to a problem that is equivalent to our formulation of the max-throughput problem. His reduction implies that if K is an optimal routing scheme for our formulation with max throughput F then, for any $F^* < F$, there is a routing scheme K^* for the queueing-theoretic formulation (where K^* is easily obtained by scaling K appropriately) with throughput F^*. We note that Kodialam's algorithm outputs a sparse routing scheme, that is, a scheme which routes tuples along at most n distinct orderings of the operators. Although our algorithm outputs a succinct representation of an optimal routing scheme from which it is possible to efficiently calculate tuple routings, the scheme itself may not be sparse.

Several other variants of the pipelined filter ordering problem have been studied recently. One such problem is as follows: given a list L of tuples and for each, the subset of filters which it satisfies, and a cost for applying each filter, find the ordering π of the filters which minimizes the sum of the costs of evaluating all tuples in L using π. This problem is NP-hard, and significant effort has been invested in development of approximation algorithms [4, 7, 10, 19].

Other recent papers have addressed on-line variants of pipelined filter ordering [15, 19]. In these settings, tuples arrive one at a time, and the operators each have an associated cost. Tuples are processed sequentially. In the standard version of the on-line problem, the goal is to minimize the ratio, over the worst-case sequence of tuples, between the cost paid on that sequence, and the cost that would have been paid if all tuples in the sequence had been processed according to the single ordering π incurring minimum total cost on this sequence. This ratio is a type of multiplicative regret, but the regret is with respect to a sequence of tuples, rather than a single tuple.

Etzioni et al. [9] studied a web-query problem with some similarities to the max-throughput problem. There are m queries and n information sources. Consulting a source has a time cost and a dollar cost, and yields the answer to a query with a certain probability. Multiple sources can be consulted at the same time. The goal is to answer all m queries while minimizing the sum of the time and dollar cost. They provide an approximation algorithm for this problem.

In *generalized maximum flow* problems, the amount of flow may change as it travels through a network (cf. Fleischer [11]). Although the flow problems studied in this paper also have this property, the requirement that flow pass through all operators (if not eliminated along the way) does not arise in generalized maximum flow problems. For some flow problems, decisions about flow routing can be made locally at nodes of the network, independently of other nodes [2]. An interesting question is whether there are efficient distributed local algorithms for the pipelined filter ordering problems of this paper.

3. THE MAX-THROUGHPUT PROBLEM

We first formally define the max-throughput problem via a linear program and present an example problem. We then give an intuitive description of our algorithm and its motivation, followed by a formal description of the algorithm. We also discuss the output format of our algorithm.

We will frequently refer to permutations of sets and introduce our notation here. Let π be a permutation of a set S of size l. We represent π as a sequence $s_1, \ldots, s_l$ of the elements of S. For $k \in [1 \ldots l]$ we use $\pi(k)$ to denote the kth element of the sequence, s_k. For $s \in S$, $\pi^{-1}(s)$ denotes the position of s in the sequence, that is, $\pi^{-1}(s) = i$ such that $s = s_i$. Suppose that π_1 and π_2 are permutations over disjoint sets S_1 and S_2. Then $\pi_1\pi_2$ denotes the permutation of $S_1 \cup S_2$ corresponding to the sequence formed by concatenating the sequences representing π_1 and π_2.

An instance of the max-throughput problem is a list of n *selectivities* (probabilities) $p_1, \ldots, p_n$, and n *rate limits* $r_1, \ldots, r_n$. The p_i are real values between 0 and 1, and the r_i are non-negative. Let $\phi(n)$ be the set of all $n!$ permutations of $\{1, \ldots, n\}$. For $j \in [1 \ldots n]$, and permutation $\pi \in \phi(n)$, let $g(\pi, j)$ denote the probability that a tuple sent according to permutation π reaches selection operator O_j without being eliminated. Thus $g(\pi, j) = p_{\pi(1)}p_{\pi(2)} \cdots p_{\pi(m-1)}$, where $m = \pi^{-1}(j)$. Define $n!$ variables f_π, one for each permutation $\pi \in \phi(n)$, where each f_π represents the number of tuples routed along permutation π per unit time. We call the f_π *flow variables*. The *max-throughput* problem is to find a solution to the following linear program. We refer to the constraints of the first type as *rate constraints*.

Max-throughput LP: Given $r_1, \ldots, r_n > 0$ and $p_1 \ldots, p_n \in [0, 1]$, maximize

$$F = \sum_{\pi \in \phi(n)} f_\pi$$

subject to the constraints

$$\sum_{\pi \in \phi(n)} f_\pi g(\pi, i) \leq r_i \text{ for all } i \in [1 \ldots n]$$

$$f_\pi \geq 0 \text{ for all } \pi \in \phi(n)$$

For example, let $n = 2$, $p_1 = p_2 = 1/2$, $r_1 = 2$, and $r_2 = 3$. If all tuples are sent to O_2 first and then to O_1, only 3 tuples per unit time can be processed. That is, if we set $f_{1,2} = 0$, then the maximum possible value of F is 3. Also, since $p_2 = 1/2$, this solution results in an expected rate of $3/2$ tuples per unit time arriving at O_1, which is below the rate limit $r_1 = 2$ of O_1. A different routing allows more tuples to be processed, namely sending $8/3$ tuples per unit time along route O_2, O_1, and $2/3$ tuples per unit time along route O_1, O_2 (i.e. $f_{2,1} = 8/3$ and $f_{1,2} = 2/3$).

3.1 Algorithm to calculate an optimal routing for the max-throughput problem

3.1.1 Introduction to the algorithm

We begin by giving an informal introduction to our routing algorithm. We view the problem of routing tuples as one of constructing a flow through the operators. The capacity of each operator is its rate limit, and the amount of flow sent along a path through the operators is equal to the number of tuples sent along that path per unit time. We treat an operator having selectivity p as outputting exactly p times the amount of flow into it, although this is actually the *expected* flow output. However, our arguments apply also to expectation.

Consider first the special case in which operators all have the same rate limit (capacity). For example, let O_1, O_2 be two operators with selectivities p_1 and p_2, and with equal rate limits r. If we send x units of flow along permutation O_1, O_2, and y units along permutation O_2, O_1, then O_1 receives $x + p_2 y$ units and O_2 receives $y + p_1 x$ units. If $x = r(1-p_2)/(2-p_1p_2)$ and $y = r(1-p_1)/(2-p_1p_2)$, then $x + p_2 y = y + p_1 x = r$ and both operators are saturated, i.e. operate at full capacity. We show that any routing which saturates all the operators achieves maximum throughput. We also give a closed-form expression that generalizes the above routing for the case of $n > 2$ operators with equal rate limits.

Now consider the case where there are n operators and all rate limits are distinct. In this case, we do not have a closed-form expression for an optimal routing. Instead, we construct a flow incrementally. Imagine pushing flow along the single permutation $O_n, \ldots, O_1$, where we have indexed the operators so that this permutation orders them in decreasing order of their rate limits. Suppose we continuously increase the amount of flow being pushed, beginning from zero, while monitoring the "residual capacity" of each operator, i.e., the difference between its rate limit and its load (the current rate of tuples arriving at that operator). For the moment, don't worry about exceeding the rate limit of an operator.

Consider two adjacent operators, O_i and O_{i-1}. As we increase the amount of flow, the residual capacity of each operator decreases continuously. Initially, at zero flow, the residual capacity of O_i is greater than the residual capacity of O_{i-1}. By continuity, the residual capacity of O_i cannot become less than the residual capacity of O_{i-1} without the two residual capacities first becoming equal. We now impose the following stopping condition: increase the flow along permutation $O_n, \ldots, O_1$ until either (1) some operator becomes saturated, or (2) the residual capacities of at least two of the operators become equal. This stopping condition ensures that when the flow increase is halted, permutation $O_n, \ldots, O_1$ orders the operators in non-increasing order of their residual capacities. (Algorithmically, we do not increase the flow continuously, but instead directly calculate the amount of flow which triggers the stopping condition.)

We show that if stopping condition (1) above holds when the flow increase is stopped, the constructed flow is optimal. If stopping condition (2) holds, we keep the current flow, and then augment it by solving a new max-throughput problem in which we set the rate limits of the operators to be equal to their residual capacities under the current flow (their selectivities remain the same).

We now generalize the case of distinct rate limits to one in which some of the rate limits of the operators are equal. Assume that permutation $O_n, \ldots, O_1$ orders the operators in non-increasing order of their rate limits. We group the operators into equivalence classes according to their rate limits. We then replace each equivalence class with a single mega-operator, with a rate limit equal to the rate limit of the constituent operators, and selectivity equal to the product of their selectivities. We then essentially apply the procedure for the case of distinct rate limits to the mega-operators. The one twist is the way in which we translate flow sent through a mega-operator into flow sent through the constituent operators of that mega-operator; we route the flow through the constituent operators so as to equalize their load. We thus ensure that the operators in each equivalence class continue to have equal residual capacity. Note that, under this scheme, the residual capacity of an operator in a mega-operator may decrease more slowly than it would if all flow were sent directly to that operator (because some flow may first be filtered through other operators in the mega-operator) and this needs to be taken into account in determining when the stopping condition is reached.

3.1.2 Example

Suppose we have 3 operators, O_3, O_2, O_1 with rate limits $r_3 = 3, r_2 = 2$, and $r_1 = 1$, and selectivities $p_1 = p_3 = 1/2$ and $p_2 = 1/4$. If we send flow along permutation O_3, O_2, O_1, then $1/2$ of it will reach O_2 and $1/8$ will reach O_1. The residual capacities of O_3 and O_2 become equal at 2 units of flow, the residual capacities of O_2 and O_1 become equal at $8/3$ units, and the minimum amount of flow needed to saturate an operator is 3 units. Therefore, the stopping condition is reached at 2 units. Thus for our initial flow, we send 2 units along permutation O_3, O_2, O_1, causing the operators to have residual capacities 1, 1, and $3/4$.

To augment this flow, we have a second stage, where we solve the problem in which O_3, O_2 and O_1 have rate limits 1, 1, and $3/4$ respectively. Replace O_3 and O_2 with a mega-operator $O_{3,2}$ having selectivity $1/2 * 1/4 = 1/8$. Consider sending flow along permutation $O_{3,2}$, O_1, dividing any flow into mega-operator $O_{3,2}$ so that $3/5$ of it is sent along permutation O_3, O_2, and $2/5$ along permutation O_2, O_3; this equalizes the load on O_3 and O_2. Under this division, t units of flow sent into $O_{3,2}$ decrease the capacity of O_2 and O_3 each by $7/10\ t$. Since the rate limits of O_2 and O_3 are 1, they therefore become saturated when $10/7$ units are sent along $O_{3,2}$, O_1. Any flow sent along $O_{3,2}$, O_1 is reduced by a factor of $1/2 * 1/4 = 1/8$ before reaching O_1, so O_1

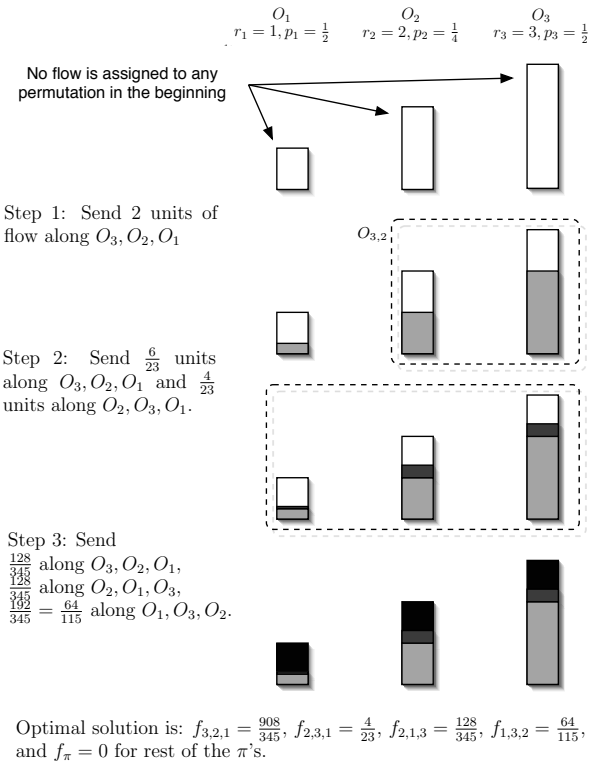

Figure 1: An example illustrating how the algorithm works

is saturated when 6 units of flow are sent along $O_{3,2}$, O_1. The residual capacities of the operators in $O_{3,2}$, and operator O_1 equalize at t units, where $1 - 7/10\ t = 3/4 - 1/8t$, that is, $t = 10/23$. Thus the stopping condition is reached at 10/23 units, when the residual capacities of the operators are equalized at 16/23. We therefore augment the initial flow with 10/23 units sent along permutation $O_{2,3}, O_1$; translated, this means 6/23 units along O_3, O_2, O_1, and 4/23 along O_2, O_3, O_1.

In the third and final stage, we solve the max-throughput problem in which operators O_3, O_2, O_1 have equal rate limits of 16/23. Using our closed-form expression (cf. Lemma 3.2) to equalize the load on the three operators, we divide the flow so 2/7 of it is sent along permutation O_3, O_2, O_1, 2/7 along permutation O_2, O_1, O_3, and 3/7 along permutation O_1, O_3, O_2. Under this division, 15/28 of the flow arrives at each operator. Thus sending $t = 448/345$ units saturates the operators, since $16/23 = 15/28t$. We augment the flow from the first two stages with 448/345 units of flow divided as just described.

Together, the flows constructed in the above three stages yield the following optimal solution to the max-throughput LP for the given input instance: $f_{3,2,1} = 908/345$, $f_{2,3,1} = 4/23$, $f_{2,1,3} = 128/345$, $f_{1,3,2} = 64/115$, and $f_\pi = 0$ for all other permutations π. This flow saturates all operators, although in general, this may not be the case. Since the number of permutations used in routing the flow may be exponential in the number of operators, our algorithm outputs a compact representation of the flow, rather than giving the values of the non-zero f_π .

3.1.3 Lemmas

We now present some technical lemmas, which will be helpful in the next section.

For any feasible solution K to the max-throughput LP, define the *residual capacity* of a selection operator O_i to be $r_i - \sum_{\pi \in \phi(n)} f_\pi g(\pi, i)$, the difference between the maximum expected rate of tuples that can be processed by O_i, and the expected rate of tuples arriving at O_i under K.

The following important lemma gives a sufficient condition for a feasible solution to be optimal.

LEMMA 3.1. *If feasible solution K to the max-throughput LP has the property that for some non-empty subset $Q \subseteq \{1 \ldots n\}$, (1) for each $i \in Q$, the rate constraint for selection operator O_i is tight, i.e. $\sum_{\pi \in \phi(n)} f_\pi g(\pi, i) = r_i$, and (2) for any flow variable f_π, $f_\pi > 0$ implies that in permutation π, the elements of $\bar{Q} = \{1 \ldots n\} \backslash Q$ precede the elements of Q, then feasible solution K is also optimal and the value F of the objective function achieved by K is*

$$\frac{\sum_{i \in Q} r_i (1 - p_i)}{(\prod_{j \notin Q} p_j)(1 - \prod_{i \in Q} p_i)}.$$

Proof. Consider the assignment to the f_π under solution K. We have that $\sum_{\pi \in \phi(n)} f_\pi g(\pi, i) = r_i$, for $i \in Q$. Multiplying both sides of this equation by $(1 - p_i)$ and summing over all $i \in Q$ we get that

$$\sum_{i \in Q} \sum_{\pi \in \phi(n)} f_\pi g(\pi, i)(1 - p_i) = \sum_{i \in Q} r_i (1 - p_i). \qquad (1)$$

We now rewrite the left hand side of Equation 1. Exchanging summation order and bringing out f_π shows it equals $\sum_{\pi \in \phi(n)} f_\pi \sum_{i \in Q} g(\pi, i)(1 - p_i)$. Let $\phi'(n)$ denote the subset of $\phi(n)$ consisting of all permutations in $\phi(n)$ in which the elements of $\bar{Q} = \{1 \ldots n\} \backslash Q$ precede the elements of Q. By assumption, $f_\pi = 0$ for all $\pi \notin \phi'(n)$. Let $\pi \in \phi'(n)$. Consider $\sum_{i \in Q} g(\pi, i)(1 - p_i)$. For all $i \in Q$, $g(\pi, i)$ equals $(\Pi_{j \notin Q} p_j)$ times the product of all p_m such that $m \in Q$ and $\pi(m) < \pi(i)$. Thus $\sum_{i \in Q} g(\pi, i)(1 - p_i) = (\Pi_{j \notin Q} p_j) \sum_{i \in Q} (\Pi_{m \in Q : \pi(m) < \pi(i)}\ p_m)(1 - p_i)$. To simplify this expression, we use the fact that

$$\sum_{i \in Q} (\Pi_{m \in Q : \pi(m) < \pi(i)}\ p_m)(1 - p_i) = (1 - \Pi_{i \in Q} p_i).$$

Hence

$$\sum_{i \in Q} g(\pi, i)(1 - p_i) = (\prod_{j \notin Q} p_j)(1 - \prod_{i \in Q} p_i)$$

and it follows [1] that

$$\sum_{\pi \in \phi(n)} f_\pi (\prod_{j \notin Q} p_j)(1 - \prod_{i \in Q} p_i) = \sum_{i \in Q} r_i (1 - p_i).$$

Thus

$$F = \sum_{\pi \in \phi(n)} f_\pi = \frac{\sum_{i \in Q} r_i (1 - p_i)}{(\prod_{j \notin Q} p_j)(1 - \prod_{i \in Q} p_i)}.$$

[1] In fact, the correctness of the next equation can also be shown as follows. For $i \in Q$, since the rate constraint for i is tight, the expected number of tuples processed by O_i per unit time is r_i, and the expected number eliminated is $r_i(1 - p_i)$. Thus the two sides of the equation both express the total expected number of tuples per unit time eliminated by operators O_i where $i \in Q$.

We now show that this value of F is as large as possible. Consider a modified version of the max-throughput LP in which we eliminate all rate constraints for operators O_j such that $j \notin Q$. Consider an optimal solution to this modified LP which assigns values f'_π to each of the variables f_π. Let $F' = \sum_{\pi \in \phi(n)} f'_\pi$. Clearly F' is an upper bound on the maximum value of the objective function for the original max-throughput LP. Let $\phi'(n)$ be as defined above. Because operators O_j such that $j \notin Q$ have no rate constraints, and because each such operator may eliminate tuples, we may assume without loss of generality that if $f'_\pi > 0$, then $\pi \in \phi'(n)$. For $i \in Q$, if we multiply both sides of the rate constraint for O_i by $(1 - p_i)$ and sum over all $i \in Q$, we get that $\sum_{\pi \in \phi'(n)} f'_\pi g(\pi, i)(1 - p_i) \leq r_i(1 - p_i)$. The same argument as above shows that the left hand side of this equation is equal to $\sum_{\pi \in \phi'(n)} f'_\pi (\prod_{j \notin Q} p_j)(1 - \prod_{i \in Q} p_i)$. It follows that $F' \leq \frac{\sum_{i \in Q} r_i(1-p_i)}{(\prod_{j \notin Q} p_j)(1-\prod_{i \in Q} p_i)}$, and thus for the original max-throughput LP, the value of the objective function cannot exceed this value. □

The following lemma gives the closed-form expression for a routing that equalizes the load on n operators.

LEMMA 3.2. *Let ρ_1 be the permutation $1, \ldots, n$ and for $j \in [2 \ldots n]$, let ρ_j be permutation $j, j+1, \ldots, n, 1, 2, \ldots, j-1$, that is, the permutation obtained by performing $j-1$ left cyclic shifts on ρ_1. Let $t > 0$. Let*

$$f_{\rho_j} = \frac{1 - p_{j-1}}{n - \sum_{k=1}^{n} p_k} t \text{ for all } j \in [1 \ldots n].$$

(where $p_0 = p_n$) and let $f_\pi = 0$ for all other $\pi \in \phi(n)$. Then $\sum_{\pi \in \phi(n)} f_\pi = t$ and for all $i \in [1 \ldots n]$,

$$\sum_{\pi \in \phi(n)} f_\pi g(\pi, i) = \frac{1 - \prod_{k=1}^{n} p_k}{n - \sum_{k=1}^{n} p_k} t. \tag{2}$$

Proof. Clearly $\sum_{\pi \in \phi(n)} f_\pi = t$. Let i be in the range $[1 \ldots n]$. For the given assignment to the flow variables, $\sum_{\pi \in \phi(n)} f_\pi g(\pi, i) = \sum_{j=1}^{n} \frac{(1-p_{j-1})}{n - \sum_{k=1}^{n} p_k} g(\rho_j, i)t$. Recall that by definition, $g(\rho_j, i) = p_j \ldots p_{i-1}$ when $j \leq i$ and $g(\rho_j, i) = p_j \ldots p_n p_1 \ldots p_{i-1}$ when $j > i$. Thus, by cancellation of terms,

$$\sum_{j=1}^{n} \frac{(1 - p_{j-1})}{n - \sum_{k=1}^{n} p_k} g(\rho_j, i)t = \frac{1}{n - \sum_{k=1}^{n} p_k} t - \frac{\prod_{k=1}^{n} p_k}{n - \sum_{k=1}^{n} p_k} t.$$

□

In our max-throughput algorithm, the selection operators are partitioned into sets $E_m, \ldots, E_1$, and we use a routing that obeys two properties. First, it only sends tuples via permutations in which the tuples travel first through the operators in E_m, then through operators in E_{m-1}, then through the operators in E_{m-2}, and so on. Second, for any set E_i in the partition, the expected tuple arrival rate is the same for all the operators in E_i. Our algorithm is based on the following technical lemma (which follows from Lemma 3.2).

LEMMA 3.3. *Let $E_m, \ldots, E_1$ be a partition of the set of operators $\{O_n, \ldots, O_1\}$ such that for $i \in [1 \ldots m]$, there exist indices $b(i) \leq c(i)$ such that $E_i = \{O_{c(i)}, O_{c(i)-1}, \ldots, O_{b(i)}\}$. Let $\beta(i)$ be the set of $|E_i| = c(i) - b(i) + 1$ permutations which are all cyclic shifts of the permutation $c(i), \ldots, b(i)$. Let P be the set of $\prod_{l=1}^{m} |E_l|$ permutations $\pi_m \ldots \pi_1$ where each $\pi_i \in \beta(i)$. Suppose we probabilistically route a total of t tuples through the operators, dividing them among the permutations in P, such that the expected rate of tuples routed via permutation $\pi_m \ldots \pi_1$ is*

$$\frac{\prod_{l=1}^{m}(1 - p_{z(\pi_l)})}{\sum_{\pi_m \ldots \pi_1 \in P} \prod_{l=1}^{m}(1 - p_{z(\pi_l)})} t$$

where $z(\pi_l)$ is the last element of the permutation π_l. Then for all $i \in [1 \ldots m]$, the expected rate of tuples arriving at any operator in E_i is $t\xi(i)$ tuples per unit time, where

$$\xi(i) = p_n p_{n-1} \ldots p_{c(i)+1} \frac{1 - \prod_{j=b(i)}^{c(i)} p_j}{|E_i| - \sum_{j=b(i)}^{c(i)} p_j}. \tag{3}$$

3.1.4 Algorithm description

We now present the routing algorithm. Without loss of generality, assume that $r_n \geq r_{n-1} \geq \ldots \geq r_1$. We describe the algorithm recursively, and $r_n \geq r_{n-1} \geq \ldots \geq r_1$ holds in the recursive calls also.

We partition the operators into equivalence classes $E_m, \ldots, E_1$, where operators are in the same class if they have the same rate limit. Denote the rate limits of the operators in $E_m, \ldots, E_1$ by $R_m, \ldots, R_1$ respectively. Assume the E_i satisfy $R_m > R_{m-1} > \ldots > R_1$.

We will use the following notation. Suppose that we send tuples through the system at a rate of t tuples per unit time, according to the method of Lemma 3.3. Then for every E_i, tuples arrive at each operator in E_i at a rate of $t\xi(i)$ tuples per unit time (where $\xi(i)$ is as defined in Equation 3 of Lemma 3.3). Let $R'_m = R'_m(t), \ldots, R'_1 = R'_1(t)$ denote the residual capacities of the operators in $E_m, E_{m-1}, \ldots, E_1$ respectively, and let $r'_n = r'_n(t), \ldots, r'_1 = r'_1(t)$ denote the residual capacities of the individual operators $O_n, \ldots, O_1$. Then at $t = 0$, $R'_m > \ldots > R'_1$ and $r'_n \geq \ldots \geq r'_1$. As t increases, each R'_i (and r'_j) decreases continuously.

Next, we set $\hat{t}$ to be the smaller of (1) $\frac{r_1}{\xi(1)}$ or (2) the minimum of $\frac{R_i - R_{i-1}}{\xi(i) - \xi(i-1)}$, taken over all $i \in [2 \ldots m]$, where ξ is as defined in Equation 3 of Lemma 3.3. The first quantity is the smallest (positive) value of t at which the residual capacity of O_1 becomes 0, and the second quantity denotes the value of t at which the values R'_i and R'_{i-1} become equal. Thus $\hat{t}$ is the value of t that meets the stopping condition described in Section 3.1.1. Let $\hat{R}_m, \ldots, \hat{R}_1$ and $\hat{r}_n, \ldots, \hat{r}_1$ denote the values of $R'_m, \ldots, R'_1$ and $r'_n, \ldots, r'_1$ respectively when $t = \hat{t}$.

We claim that $\hat{R}_m \geq \ldots \geq \hat{R}_1$. Suppose not. Then $\hat{R}_i < \hat{R}_{i-1}$ for some i. Since at $t = 0$, $R'_i > R'_{i-1}$, both quantities decrease continuously as t increases, and $R'_i < R'_{i-1}$ at $t = \hat{t}$, there must be a value of t that is less than $\hat{t}$ for which $R'_i = R'_{i-1}$. But this contradicts our choice of $\hat{t}$. We have thus shown that $\hat{R}_m \geq \ldots \geq \hat{R}_1$ and hence $\hat{r}_n \geq \ldots \geq \hat{r}_1$.

Let K be the assignment to the flow variables induced by routing $\hat{t}$ tuples per unit time according to Lemma 3.3. To do our computation in polynomial time, we represent K succinctly, as the pair consisting of the partition $E_m, \ldots, E_1$ and the value $\hat{t}$ (from which, using Lemma 3.3, we can determine K).

If $\hat{t}$ equals quantity (1), namely $\frac{r_1}{\xi(1)}$, then we output K.

Otherwise, it must be that (2) < (1), and we recursively run the algorithm with selectivities $p_n, \ldots, p_1$ and rate limits $\hat{r}_n, \ldots, \hat{r}_1$. Note that for any j, k, if $r_j = r_k$, then $\hat{r}_j = \hat{r}_k$. Further, for at least one j, $r_j \neq r_{j+1}$, but $\hat{r}_j = \hat{r}_{j+1}$. Thus the equivalence classes E_i in each recursive call are formed by merging equivalence classes from the previous call, and the total number of equivalence classes decreases in each recursive call.

Let K' be the solution returned by the recursive call. We output K'', the solution to the LP which is obtained by setting each flow variable f_π to the sum of its value in K and K'. We can represent K'' succinctly as the concatenation of the representations of K and K'.

This completes the description of the algorithm. The number of equivalence classes decreases in each recursive call, so the number of recursive calls is at most $n-1$. The time per recursive call is $O(n)$. Therefore, the algorithm runs in time $O(n^2)$.

It remains to prove that the algorithm outputs an optimal solution to the max-throughput LP. In the final recursive call, since $\hat{R}_m \geq \ldots \geq \hat{R}_1$, there is a maximum i such that $\hat{R}_i = \hat{R}_{i-1} = \ldots = \hat{R}_1 = 0$, and no other $\hat{R}_j$ is equal to 0. Let $O_q, O_{q-1}, \ldots, O_1$ be the operators in $E_i, E_{i-1}, \ldots, E_1$. Let $Q = \{O_q, O_{q-1}, \ldots, O_1\}$. Then in the final solution to the original max-throughput problem, constructed from all the recursive calls, Q is the set of operators with residual capacity 0. Also, since the partitions in each recursive call are formed by merging sets of the partition in the previous call, tuples are only routed along permutations in which the elements in Q appear at the end (in some order). It follows from Lemma 3.1 that the output solution is optimal.

The output of the algorithm is a list of some n' pairs $(P_1, \hat{t}_1), \ldots, (P_{n'}, \hat{t}_{n'})$, where the P_i's are the partitions of operators and the t_i's are the $\hat{t}$ values. To use this representation in order to actually route tuples in a distributed environment, first calculate the sum $T = \sum_{i=1}^{n'} \hat{t}_i$. Send T tuples per unit time using the following procedure to route each tuple. For each tuple, first randomly choose an $i \in [1 \ldots n']$, with probability proportional to $\hat{t}_i$. For the chosen i, suppose $P_i = E_m, \ldots, E_1$. For each $j \in [1 \ldots m]$, randomly choose a permutation π_j from the permutations in $\beta(j)$ (defined in Lemma 3.3), with probability proportional to $1 - p_{z(\pi_j)}$, where $z(\pi_j)$ is the last element of the ordering π_j. Then route the tuple via permutation $\pi_m \ldots \pi_1$.

3.2 Algorithm to compute the value of the max throughput

Given an instance of the max-throughput problem with the rates $r_1, r_2, \ldots, r_n$ in sorted order, the value of the max throughput can be calculated in linear time, as follows. For each $q \in [1, \ldots, n]$, let

$$F_q = \frac{\sum_{i=1}^{q} r_i(1-p_i)}{(\prod_{j=q+1}^{n} p_j)(1 - \prod_{i=1}^{q} p_i)}.$$

The algorithm simply calculates $F_1, F_2, \ldots F_n$ in turn, and outputs the maximum of these values. Each value F_{q+1} can be calculated in constant time, given the numerator and denominator of F_q.

The correctness of this algorithm follows directly from the construction of an optimal routing given in Section 3.1.4 and its proof of correctness. Specifically, let K be the optimal routing of our construction. Then for some $q \in [1, \ldots, n]$, the set $Q = \{1, \ldots, q\}$ witnesses the fact that K satisfies the property given in the statement of Lemma 3.1. From Lemma 3.1 it follows that value F_q is the value F of the max throughput achieved by K.

4. THE GAME THEORETIC MULTIPLICATIVE REGRET (GTMR) PROBLEM

We begin by giving a formal definition of this problem. An instance of the GTMR problem is a list of positive real costs $c_1, \ldots, c_n$. Let $h(\pi, i) = c_{\pi(1)} + c_{\pi(2)} + \ldots + c_{\pi(m)}$ where $m = \pi^{-1}(i)$. Let $\phi(n)$ denote the set of all permutations of $\{1, \ldots, n\}$. The GTMR problem is given by the minimax formulation below. The f_π denote the probability of choosing to route the tuple through the operators according to the ordering specified by permutation π.

Game theoretic multiplicative regret:
Given $c_1, \ldots, c_n > 0$, minimize

$$\max_{i \in [1 \ldots n]} \sum_{\pi \in \phi(n)} f_\pi h(\pi, i)/c_i$$

subject to the constraints

$$\sum_{\pi \in \phi(n)} f_\pi = 1$$

$$f_\pi \geq 0 \text{ for all } \pi \in \phi(n)$$

For example, consider an instance of the GTMR problem with $c_1 = c_2 = c_3$. The intuitive strategy of choosing a random routing (uniformly) is optimal. An alternative optimal strategy is to choose the orderings 1,2,3; 2,3,1; and 3,1,2 with equal probability.

A contrasting example is when the costs are $c_1 = 2, c_2 = 2$, and $c_3 = 8$. If the adversary selects O_1 or O_2 to eliminate the tuple, then routing the tuple to O_3 first is bad – it results in an expected multiplicative regret of at least 5 $(= (8+2)/2)$. In fact, it can be shown that the only optimal strategy for the routing player is to choose one of the orderings 1,2,3 and 2,1,3, each with probability 1/2, yielding expected multiplicative regret of 3/2. Note that both these orderings have 3 in the last position. Finally, suppose $c_1 = c_2 = 2$ and $c_3 = 7$. In this case, one optimal strategy is to choose from the permutations 1,2,3; 1,3,2; and 2,1,3, with probabilities 23/57, 2/57, and 32/57, respectively.

4.1 Algorithm to calculate an optimal routing for the GTMR problem

We relate the max-throughput problem and the GTMR problem by studying an (artificial) problem that we call the *cumulative cost limit problem*. The solution to this problem has many similarities to the solution to the max-throughput problem.

In the cumulative cost limit problem, we again have n operators $O_1, O_2, \ldots, O_n$ with costs, and we need to decide how many tuples per unit time to route along each permutation. However, in this problem there is also a cost limit d_i associated with each operator O_i. Tuples cannot be eliminated by

operators, and the processing of each tuple is deterministic. Costs are cumulative, so that when a tuple arrives at an operator O_i, the amount that must be paid for O_i to process it is c_i plus the sum of all costs c_j associated with operators O_j that have already processed that tuple. Operators have no limit on the *number* of tuples they can process per unit time. Instead, they are limited by their cumulative cost limit d_i, which is an upper bound on the total amount that can be paid for using that operator per unit time. The problem is to route the tuples so as to maximize the rate of tuples that can be processed, subject to the cumulative cost limits.

Formally, the *cumulative cost limit problem* is given by the linear program below, where $\phi(n)$ denotes the set of permutations of $\{1,\ldots,n\}$ as before, and $h(\pi,i) = \sum_{j=1}^{m} c_{\pi(j)}$, where $m = \pi^{-1}(i)$.

Cumulative cost limit LP: Given $c_1,\ldots,c_n > 0$ and $d_1,\ldots,d_n > 0$, maximize

$$F = \sum_{\pi\in\phi(n)} f_\pi$$

subject to the constraints

$$\sum_{\pi\in\phi(n)} f_\pi h(\pi,i) \le d_i \text{ for all } i \in [1\ldots n]$$

$$f_\pi \ge 0 \text{ for } \pi \in \phi(n)$$

The following lemma is analogous to Lemma 3.1.

LEMMA 4.1. *If a feasible solution to the cumulative cost limit LP has the property that for some non-empty subset $Q \subseteq \{1,\ldots,n\}$, (1) for each $i \in Q$, the cumulative cost limit constraint for operator O_i is tight and (2) for any flow variable f_π, $f_\pi > 0$ implies that in permutation π, the elements of Q precede the elements of $\bar{Q}$, then the feasible solution is optimal and the value of the objective function under the feasible solution is*

$$\frac{\sum_{i\in Q} c_i d_i}{(\sum_{i\in Q} c_i^2) + (\sum_{i,j\in Q, i<j} c_i c_j)} \tag{4}$$

Proof. Consider a feasible solution satisfying the conditions of the lemma. It specifies the rate at which tuples should be sent along each permutation. Let $C_Q = \sum_{j\in Q} c_j$. For each $i \in Q$, the cumulative cost limit constraint for O_i is tight, i.e. $\sum_{\pi\in\phi(n)} f_\pi h(\pi,i) = d_i$. Multiplying both sides of this constraint by $\frac{c_i}{C_Q}$ and summing over all $i \in Q$, we get that

$$\sum_{i\in Q}\sum_{\pi\in\phi(n)} f_\pi h(\pi,i)\frac{c_i}{C_Q} = \sum_{i\in Q}\frac{d_i c_i}{C_Q}. \tag{5}$$

Exchanging the order of the summations on the left hand side of the equation shows it is equal to

$$\sum_{\pi\in\phi(n)} f_\pi \sum_{i\in Q} h(\pi,i)\frac{c_i}{C_Q}. \tag{6}$$

Let $\phi'(n)$ denote the permutations of $\phi(n)$ in which the elements of Q precede the elements of $\bar{Q}$. By assumption, $f_\pi = 0$ for all $\pi \notin \phi'(n)$. Let $\pi \in \phi'(n)$. Consider $\sum_{i\in Q} h(\pi,i)\frac{c_i}{C_Q}$. The quantity $h(\pi,i)$ is equal to c_i plus the sum of the c_j such that j precedes i in π. For any $j,k \in Q$ such that $j \ne k$, either j precedes k in π and c_j is an element of the sum $h(\pi,k)$, or k precedes j in π, and c_k is an element of the sum $h(\pi,j)$. It follows that $\sum_{i\in Q} h(\pi,i)\frac{c_i}{C_Q} = U_Q/C_Q$, where $U_Q = (\sum_{i\in Q} c_i^2) + (\sum_{i,j\in Q, i<j} c_i c_j)$. Hence $\sum_{\pi\in\phi(n)} f_\pi U_Q/C_Q = \sum_{i\in Q}\frac{d_i c_i}{C_Q}$. It immediately follows that $F = \sum_{\pi\in\phi(n)} f_\pi = \frac{\sum_{i\in Q} d_i c_i}{U_Q}$.

We now show that the value of F cannot be larger than this value, for any feasible solution to the cumulative cost limit LP. Consider a modified version of the cumulative cost limit LP in which we eliminate all cost limit constraints for selection operators O_j such that $j \notin Q$. Consider an optimal solution to this modified problem which assigns values f'_π to each of the variables f_π. Let $F' = \sum_{\pi\in\phi(n)} f'_\pi$ be the value of the objective function. Clearly F' is an upper bound on the maximum possible value of the objective function for the original cumulative cost limit LP. Let $\phi'(n)$ be as defined above. Because selection operators O_j such that $j \notin Q$ have no cost limit constraints, and because each operator can only increase the cumulative amount of cost that will be passed on to subsequent operators, we may assume without loss of generality that if $f'_\pi > 0$, then $\pi \in \phi'(n)$. If we take the cost limit constraints for O_i where $i \in Q$, multiply both sides of each by $\frac{c_i}{C_Q}$, and add the resulting inequalities, we get that $\sum_{i\in Q}\sum_{\pi\in\phi(n)} f'_\pi h(\pi,i)\frac{c_i}{C_Q} \le \sum_{i\in Q}\frac{d_i c_i}{C_Q}$. The same argument as above shows that $F' \le \frac{\sum_{i\in q} d_i c_i}{U_Q}$. □

We now show how to route t tuples per unit time so as to ensure that each operator has the same cumulative cost per unit time.

LEMMA 4.2. *Let ρ_1 be the permutation $1,\ldots,n$ and for $j \in [2\ldots n]$, let ρ_j be permutation $j, j+1,\ldots,n,1,2,\ldots,j-1$, that is, the permutation obtained by performing $j-1$ left cyclic shifts on ρ_1. Let $t > 0$. Suppose we send a total of t tuples per unit time through the operators, using the routing which sets f_{ρ_i} to $t\frac{c_i}{\sum_{j=1}^n c_j}$ for all $i \in [1\ldots n]$, and sets $f_\pi = 0$ for all other $\pi \in \phi(n)$. Then the amount that must be paid for every operator per unit time is $\frac{(\sum_{j=1}^n c_j^2)+(\sum_{1\le i<j\le n} c_i c_j)}{\sum_{j=1}^n c_j} t$.*

Proof. Let $i \in [1\ldots n]$. The quantity $\sum_{\pi\in\phi(n)} f_\pi h(\pi,i)$ is equal to
$t\sum_{j=1}^n \frac{c_j}{\sum_{k=1}^n c_k} h(\rho_j,i)$, where $h(\rho_j,i)$ is equal to c_i plus the sum of the c_k such that k precedes i in ρ_j. Expanding the term $h(\rho_j,i)$ and multiplying out, each permutation ρ_j contributes to the expression the term $(c_j^2/\sum_{j=1}^n c_j)t$ and terms $(c_j c_k/\sum_{j=1}^n c_j)t$ for all k such that k precedes i in ρ_j. Consider any $j,k \in [1\ldots n]$ such that $j < k$ and $j,k \ne i$. If $j < i < k$, then j precedes i in ρ_k but k does not precede i in ρ_j. If, on the other hand, $i < j$ or $i > k$, then k precedes i in ρ_j, but j does not precede i in ρ_k. It follows that the sum of the terms of the expression is $t\frac{(\sum_{j=1}^n c_j^2)+(\sum_{1\le i<j\le n} c_i c_j)}{\sum_{j=1}^n c_j}$. □

As in the max-throughput problem, we use this lemma to prove a technical lemma that gives a method of routing tuples so that, given a partition $E_m,\ldots,E_1$, we only send tuples along permutations in which operators in E_m are first,

E_{m-1} are next, and so on, and such that the cumulative cost per unit time for an operator to process the tuples within any given set in the partition is the same for all operators.

With the technical lemma, we have the same building blocks that we had for the max-throughput algorithm, and we can essentially run the same algorithm to solve the cumulative cost limit problem (with the routing method for keeping costs equal, a different calculation for computing $\hat{t}$, and with the operators ordered in increasing cost limit order, rather than in decreasing rate limit order). Lemma 4.1 proves that the solution computed by the algorithm is optimal.

In standard routing problems with limits on the capacity of edges (or nodes), congestion minimization and throughput maximization are closely related. Congestion is the maximum, over all edges, of the *relative load* of an edge, the amount of flow through the edge divided by the capacity of the edge. If there is a routing of k flow units that achieves 5% congestion, then scaling the 5% congestion routing by a factor of 20 yields throughput of $20k$ (with 100% congestion). Comparison of the GTMR LP to the cost limit LP reveals that the GTMR problem is the congestion minimization problem corresponding to the cumulative cost limit (max-throughput) problem, and flow achieving minimimum (cost) congestion can be scaled to achieve maximum throughput. In the next lemma, we formally reduce the GTMR problem to the cumulative cost limit problem. In what follows, for any assignment A of values to the flow variables f_π, $\pi \in \phi(n)$, let $f_\pi(A)$ denote the value assigned to f_π by A.

LEMMA 4.3. *Let I_{mult} be an instance of the GTMR problem with costs $c_1, \ldots, c_n > 0$. Let I_{cost} be the instance of the cumulative cost limit problem with costs $c_1, c_2, \ldots, c_n$ and cumulative cost limits $d_1 = c_1, d_2 = c_2, \ldots, d_n = c_n$. Let K be the optimal solution to I_{cost}, and let F be the value of the objective function achieved by K. Let L be the assignment to flow variables f_π such that for each $\pi \in \phi(n)$, $f_\pi(L) = f_\pi(K)/F$. Then L is an optimal solution for I_{mult}.*

Proof. Since K is an optimal solution to I_{cost}, F is the maximum value of the objective function for I_{cost}. We show that L is an optimal solution for I_{mult}.

Since $\sum_{\pi\in\phi(n)} f_\pi(K) = F$, $\sum_{\pi\in\phi(n)} f_\pi(K)/F = 1$. Hence L satisfies the constraints of the GTMR problem. Since K maximizes the value of the objective function for the instance I_{cost} of the cumulative cost limit problem, there must be at least one i such that $\sum_{\pi\in\phi(n)} f_\pi(K)h(\pi,i) = c_i$ and hence $\sum_{\pi\in\phi(n)}(f_\pi(K)/F)h(\pi,i)/c_i = \frac{1}{F}$. Also, for every i, $\sum_{\pi\in\phi(n)}(f_\pi(K)/F)h(\pi,i)/c_i \leq \frac{1}{F}$.

Let H be the value of the objective function achieved by L for problem I_{mult}. That is, H is the maximum of $\sum_{\pi\in\phi(n)}(f_\pi(K)/F)h(\pi,i)/c_i$ over all i. Thus $H = \frac{1}{F}$.

Suppose L is not an optimal solution to the instance I_{mult} of the GTMR problem. Then there exists some other solution $\tilde{L}$ that is optimal. Let $\tilde{H}$ be the value of the objective function achieved by $\tilde{L}$. Thus $\tilde{H} < H$.

Let $\tilde{K}$ be the assignment to the flow variables such that $f_\pi(\tilde{K}) = Ff_\pi(\tilde{L})$ for all $\pi \in \phi(n)$. The value of the objective function for I_{cost} achieved by $\tilde{K}$ is $\sum_{\pi\in\phi(n)} f_\pi(\tilde{K}) = \sum_{\pi\in\phi(n)} Ff_\pi(\tilde{L}) = F$ because $\sum_{\pi\in\phi(n)} f_\pi(\tilde{L}) = 1$.

Since $\tilde{H} < H$, under solution $\tilde{L}$, for all i,

$$\sum_{\pi\in\phi(n)} f_\pi(\tilde{L})h(\pi,i)/c_i \leq \tilde{H} < H = \frac{1}{F},$$

and thus $\sum_{\pi\in\phi(n)} Ff_\pi(\tilde{L})h(\pi,i) < c_i$.

Therefore, $\sum_{\pi\in\phi(n)} f_\pi(\tilde{K})h(\pi,i) < c_i$. That is, $\tilde{K}$ is a feasible solution to I_{cost} such that none of the constraints are tight. It follows that there is a feasible solution $\tilde{M}$ to I_{cost} such that the value of the objective function under $\tilde{M}$ is greater than F. But this contradicts that F is the maximum possible value of the objective function for I_{cost}. □

The above reduction, together with the algorithm for the cumulative cost limit problem, yield an $O(n^2)$ algorithm for solving the GTMR problem.

4.2 Comparison of naive vs. optimal strategies for the GTMR problem

Consider the GTMR problem, with costs $0 < c_1 \leq \ldots \leq c_n$. Let *Naive* be the deterministic strategy for the routing player that orders the operators in increasing order of their costs. Let *Opt* be the optimal strategy for the routing player that we obtain in Lemma 4.3. Let $v_{GTMR}(Naive)$ be the expected multiplicative regret when the routing player uses strategy *Naive* and the adversary uses the best (possibly randomized) strategy against *Naive*. Similarly, let $v_{GTMR}(Opt)$ be the expected multiplicative regret when the routing player uses *Opt* and the adversary uses the best (possibly randomized) strategy against *Opt*. Note that

$$v_{GTMR}(Naive) = \max_k \frac{\sum_{i=1}^k c_i}{c_k}.$$

This is because $\frac{\sum_{i=1}^k c_i}{c_k}$ is the multiplicative regret when the routing player uses *Naive* and the adversary chooses operator k to discard the tuple; thus the best strategy of the adversary against *Naive* maximizes this value.

Also, it is the case that

$$v_{GTMR}(Opt) = \max_k \frac{\sum_{i=1}^k c_i^2 + \sum_{1\leq i<j\leq k} c_i c_j}{\sum_{i=1}^k c_i^2}. \quad (7)$$

Briefly, this follows from two inequalities. To obtain the first, consider the strategy of the adversary that chooses O_i (to be the operator which discards the tuple) with probability $\frac{c_i^2}{\sum_{i=1}^k c_i^2}$ if $i \in [1 \ldots k]$ and with probability 0 otherwise. It can be shown that for any permutation π, if the routing player routes the tuple (deterministically) according to π, the expected multiplicative regret is at least $\frac{\sum_{i=1}^k c_i^2 + \sum_{1\leq i<j\leq k} c_i c_j}{\sum_{i=1}^k c_i^2}$ against this strategy of the adversary. Hence,

$$\max_k \frac{\sum_{i=1}^k c_i^2 + \sum_{1\leq i<j\leq k} c_i c_j}{\sum_{i=1}^k c_i^2} \leq v_{GTMR}(Opt).$$

Second, Lemmas 4.1 and 4.3, together with the algorithm of Lemma 4.2, imply that for some k,

$$v_{GTMR}(Opt) = \frac{\sum_{i=1}^k c_i^2}{(\sum_{i=1}^k c_i^2) + (\sum_{1\leq i<j\leq k} c_i c_j)}.$$

Hence $v_{GTMR}(Opt) \leq \max_k \frac{\sum_{i=1}^k c_i^2}{(\sum_{i=1}^k c_i^2)+(\sum_{1\leq i<j\leq k} c_i c_j)}$. Combining both inequalities yields (7).

LEMMA 4.4.

$$\frac{v_{GTMR}(Naive)}{v_{GTMR}(Opt)} \leq 2.$$

Proof. Let $m = \arg\max_k \frac{\sum_{i=1}^k c_i}{c_k}$, that is, m is the value of k that maximizes the multiplicative regret under the $Naive$ strategy. Thus $v_{GTMR}(Naive) = \frac{\sum_{i=1}^m c_i}{c_m}$. Note that

$$\begin{aligned} v_{GTMR}(Opt) &= \max_k \frac{\sum_{i=1}^k c_i^2 + \sum_{1\leq i<j\leq k} c_i c_j}{\sum_{i=1}^k c_i^2} \\ &\geq \frac{\sum_{i=1}^m c_i^2 + \sum_{1\leq i<j\leq m} c_i c_j}{\sum_{i=1}^m c_i^2} \\ &= \frac{(\sum_{i=1}^m c_i)^2 + \sum_{i=1}^m c_i^2}{2\sum_{i=1}^m c_i^2}. \end{aligned}$$

Therefore,

$$\begin{aligned} \frac{v_{GTMR}(Naive)}{v_{GTMR}(Opt)} &\leq \left(\frac{\sum_{i=1}^m c_i}{c_m}\right)\left(\frac{2\sum_{i=1}^m c_i^2}{(\sum_{i=1}^m c_i)^2 + \sum_{i=1}^m c_i^2}\right) \\ &\leq \left(\frac{\sum_{i=1}^m c_i}{c_m}\right)\left(\frac{2c_m\sum_{i=1}^m c_i}{(\sum_{i=1}^m c_i)^2}\right) = 2. \end{aligned}$$

□

Acknowledgments We thank Joseph M. Hellerstein for illuminating discussions and for initiating this work. We thank Boris Aronov, Richard Van Slyke, and Shiyan Hu for their insights and for a conjecture that led to a linear-time algorithm for the dual of the GTMR problem. We thank Kamesh Munagala for useful background information and Rachel Pottinger for valuable feedback on an earlier draft.

5. REFERENCES

[1] R. Avnur and J. M. Hellerstein. Eddies: Continuously adaptive query processing. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 261–272. ACM Press, 2000.

[2] B. Awerbuch and F. T. Leighton. A simple local-control approximation algorithm for multicommodity flow. In *Proceedings of the 34th IEEE Symposium on Foundations of Computer Science*, pages 459–468. IEEE Computer Society, 1993.

[3] S. Babu, R. Motwani, K. Munagala, I. Nishizawa, and J. Widom. Adaptive ordering of pipelined stream filters. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, pages 407–418. ACM Press, 2004.

[4] A. Bar-Noy, M. Bellare, M.M. Halldórsson, H. Shachnai, and T. Tamir. On chromatic sums and distributed resource allocation. *Inform. and Comput.*, 140(2):183–202, 1998.

[5] S. Chaudhuri, U. Dayal, and T. W. Yan. Join queries with external text sources: Execution and optimization techniques. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 410–422. ACM Press, 1995.

[6] E. Coffman and I. Mitrani. A characterization of waiting time performance realizable by single server queues. *Operations Research*, 20:810–821, 1980.

[7] E. Cohen, A. Fiat, and H. Kaplan. Efficient sequences of trials. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 737–746. ACM Press, 2003.

[8] A. Deshpande, C. Guestrin, S. Madden, and W. Hong. Exploiting correlated attributes in acquisitional query processing. In *Proceedings of the 21st International Conference on Data Engineering (ICDE)*, pages 143–154. IEEE Computer Society, 2005.

[9] O. Etzioni, S. Hanks, T. Jiang, R. M. Karp, O. Madani, and O. Waarts. Efficient information gathering on the internet. In *Proceedings of the 37th IEEE Symposium on Foundations of Computer Science*, pages 234–243. IEEE Computer Society, 1996.

[10] U. Feige, L. Lovász, and P. Tetali. Approximating min-sum set cover. *Algorithmica*, 40(4):219–234, 2004.

[11] L. Fleischer and K. Wayne. Fast and simple approximation schemes for generalized flow. *Mathematical Programming*, 91(2):215–238, 2002.

[12] M. Garey. Optimal task scheduling with precedence constraints. *Discrete Mathematics*, 4:37–56, 1973.

[13] E. Gelenbe and I. Mitrani. *Analysis and synthesis of computer systems*. Academic Press, 1980.

[14] R. Goldman and J. Widom. Wsq/dsq: A practical approach for combined querying of databases and the web. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 285–296. ACM, 2000.

[15] H. Kaplan, E. Kushilevitz, and Y. Mansour. Learning with attribute costs. In *The Thirty-Seventh Annual ACM Symposium on Theory of Computing (STOC)*, pages 356–365. ACM Press, 2005.

[16] S. Karlin. *Mathematical Methods and Theory in Games, Programming, and Economics*. Dover, 2003.

[17] M.S. Kodialam. The throughput of sequential testing. In *Integer Programming and Combinatorial Optimization (IPCO) LNCS 2081*, pages 280–292. Springer-Verlag Berlin Heidelberg, 2001.

[18] Ravi Krishnamurthy, Haran Boral, and Carlo Zaniolo. Optimization of nonrecursive queries. In Wesley W. Chu, Georges Gardarin, Setsuo Ohsuga, and Yahiko Kambayashi, editors, *VLDB'86 Twelfth International Conference on Very Large Data Bases, August 25-28, 1986, Kyoto, Japan, Proceedings*, pages 128–137. Morgan Kaufmann, 1986.

[19] K. Munagala, S. Babu, R. Motwani, and J. Widom. The pipelined set cover problem. In *Tenth Intl. Conf. on Database Theory*, pages 83–98, 2005.

[20] M.A. Shayman and E. Fernandez-Gaucherand. Risk-sensitive decision-theoretic diagnosis. *IEEE Transactions on Automatic Control*, 46:1166–1171, 2001.

[21] H. Simon and J. Kadane. Optimal problem-solving search: All-or-none solutions. *Artificial Intelligence*, 6:235–247, 1975.