large-scale search engines [14]. These intersection lists, however, take up additional space dictating a cost-benefit trade-off, and careful strategies have been proposed to select the pairs of terms for which intersection lists should be pre-computed [6, 14].

Even though early-termination aggregation algorithms and the utility of pre-computed intersection lists have been studied extensively, there has been little work on a rigorous treatment of a combination of these two natural approaches to perform efficient top-k aggregation. Can an algorithm that uses a properly designed early termination on intersection lists significantly outperform an algorithm that is based either on an early termination on the original lists or on a complete scan of the intersection lists? And, to derive the full benefits of intersection lists, what is the principled way to perform early termination with intersection lists? Addressing these questions is precisely the focus of this paper.

Our contributions. We provide a formal analysis of two well-studied early-termination algorithms, namely, TA and NRA, in the presence of intersection lists. Specifically, we generalize both TA and NRA to the case when pre-aggregated intersection lists are present in addition to the original lists. We show that our versions of TA and NRA continue to remain instance optimal even in this case, i.e., even though the optimum algorithm has access to the intersection lists, our generalizations perform on par with the optimum. In fact, our algorithms will use *all* of the given lists. While this might appear wasteful since some of the intersections can yield redundant information, we show that this is unavoidable: Any algorithm that hopes to be instance optimal cannot afford to ignore even one of the input intersection lists. Our algorithmic generalizations also hold in the case when we can pre-compute intersections of more than two lists, and when only prefixes of pre-aggregated intersections are stored, pruned to remove low scoring elements.

The main technical challenge involved in the generalization of TA is incorporating the extra information given by the intersection lists to compute an upper bound on the aggregated score of yet unseen objects; an analogous step is needed in NRA. Unlike the case with original TA and NRA, it turns out that the upper bound here is given by a mathematical program. For simple decomposable aggregation functions such as addition, this simplifies to a linear program that can be solved in polynomial time. Addition is a natural aggregation function that is of interest in IR, where the relevance score of a document to a multi-term query is the sum of the relevance scores of the document to each of the terms in the query.

While the linear program gives the optimum upper bound, it can be expensive to solve, especially if the number of lists is large. We attack this problem in two ways. First, for the special case of the addition function and for pairwise intersections, we give a simple combinatorial algorithm based on finding the minimum cost perfect matching in a specific graph. For larger-sized intersections, we provide a simple approximation algorithm that computes an answer within a small constant factor of the optimum upper bound. In theory, instance optimality is not robust against approximate upper bounds; in practice, however, this approximation might suffice.

We conduct experiments on two collections of data: The queries provided in the TREC GOV2 dataset consisting of about 25 million documents, and query trace from the Yahoo! search engine run over an index created on 16 million randomly sampled web pages. We compare the performance of our algorithms against two baselines: One that uses intersection lists but does not terminate early and one that can terminate early but does not use intersection lists. We show that the effects of early-termination and intersection lists are synergistic. On realistic query traces, we show that the practical benefits of intersection lists can be fully realized only with early-termination algorithms.

Organization of the paper. The paper is organized as follows. Section 2 discusses related work in the areas of top-k aggregation and pre-aggregating intersection lists. Section 3 contains our main results, including our generalizations of TA and NRA, the instance optimality of the generalizations, the mathematical program to compute the upper bounds, the combinatorial matching-based algorithm, and the approximation algorithm for addition. Section 4 contains the experimental results on the TREC GOV2 and a web search index dataset. Concluding remarks are given in Section 5.

2. RELATED WORK

The related work falls into two main categories: Top-k processing in databases and IR and the use of intersections to speed up top-k processing, especially in the context of web IR.

There have been several algorithms proposed in the database and IR communities for efficient computation of top k results using pruning or early-termination methods. In the context of aggregating ranked lists with scores, the work of Fagin [7], which introduced Fagin's Algorithm (FA) and the work of Fagin, Lotem, and Naor [9], which introduced the Threshold Algorithm (TA) and the No Random-access Algorithm (NRA), are significant milestones. For a general description of these, see the survey by Fagin [8]. The TA algorithm was independently discovered by (at least) two other groups: Nepal and Ramakrishna [16] and Günter, Balke, and Kiessling [10]. Instance optimality was introduced in [9] and the algorithms TA and NRA were shown to satisfy this notion. Subsequent to their development, the FA, TA, and NRA algorithms have found applications in other areas including query processing in distributed and P2P environments [26] and integrating structure indexes and inverted lists [12]. However, none of these considered analogs of FA, TA, or NRA algorithms for intersection lists.

The work that is closest to ours is that of Das et al. in [6], who considered the problem of answering top-k queries using views, where a view is a materialized version of a list that ranks values according to a positive linear combination of a subset of attributes of a relation. They proposed an LP-based algorithm that generalizes TA to this setting. Indeed, while their work addresses a more general version of our problem, they neither show instance optimality of the resulting algorithms, nor consider the more realistic NRA in their setting, nor give combinatorial algorithms for the problem, relying instead on generic LP solvers

The field of IR has also considered early-termination algorithms for processing inverted lists. Earlier work include [5, 11, 22, 25]. Persin, Zobel, and Sacks-Davis [18] proposed the use of frequency sorting (as opposed to sorting by document ID) in posting lists in conjunction with simple early-termination algorithms. Anh, Kretser, and Moffat [1] developed a new inverted file structure using quantized weights that provides cost-effective searching when early-termination heuristics are employed. Anh and Moffat [2, 3] developed methods to reduce the number of candidate documents considered during the evaluation of ranked queries. Long and Suel [13] proposed a pruning technique that integrates hyperlink information with standard term-based information. Soffer et al. [21] considered an extreme case of efficient top-k processing, where low-scoring postings are completely eliminated from the index, and show that this can be done with very little loss in precision. Ntoulas and Cho [17] studied how to avoid degradation of result quality due to pruning while still realizing most of its benefit.

The use of intersections or pairs of terms to improve query processing has been addressed in several papers. Long and Suel [14] considered a three-level caching scheme for improving search engine performance, where the intermediate level is tasked to exploit frequently occurring pairs of terms by caching intersections or pro-

jections of the corresponding inverted lists. A similar scheme was proposed in the context of P2P-based search [4]. In [14], it is mentioned in passing that it is possible to combine pruning and list intersections; but this issue was not investigated in detail. Building intersections of posting lists is also related to building optimized indexed structures to take commonly occurring phrase queries, e.g., "new york", into account; see, for example, [23]. The problem of distributing posting lists via careful assignment of inverted lists to machines in order to reduce intersection costs was studied by Zhang and Suel [27]. Schenkel et al. [19] proposed a precomputed and materialized index structure and integrate it with a top-k query engine for proximity search.

3. MAIN RESULTS

In this section we highlight the challenges introduced by the integration of intersection lists into the TA and NRA algorithms, and present our formal results. Since our results apply to any pre-aggregation of lists, for sake of clarity, we use the term *combination* instead of intersection. The additional information provided makes the computation of the stopping condition for TA and NRA a non-trivial endeavor, and we explore both optimal and approximation algorithms and remark on their impact on instance optimality for both of these approaches.

3.1 Model

We adopt the notation and conventions established previously by Fagin and others [7, 9]. In the classic scenario the database $\mathbf{D}$ contains a set $\mathcal{X}$ of n *objects* where each object $X \in \mathcal{X}$ has m different numeric scores $(x_1, \ldots, x_m)$, which we will call *parameters*. We can think of the database as consisting of m sorted lists, $L_1, \ldots, L_m$, and each element in list L_i has a pair (X, x_i) where x_i is the i-th score of X. Each list L_i is stored in (decreasing) sorted order by the x_i's. We use $[m] = \{1, \ldots, m\}$.

In this work we investigate the effect of having precomputed some list combinations. For a given $S \subseteq [m]$, let the list L_S be composed of the combination of lists $\{L_i\}_{i \in S}$.[1] Our algorithms work in the *limited information* case, where each element of L_S is of the form $(X, f_S(\{x_j \mid j \in S\}))$, where f_S is a partial aggregation function. In this case we do *not* learn the individual scores of X, but only learn the partially aggregate score. The results carry over trivially to the *full information* case where in addition to knowing the partially aggregated score, we also learn the individual scores $\{x_j\}_{j \in S}$ of X.

We are interested in retrieving the top k elements under some aggregation function h. We make a series of assumptions about h.

(A1) Monotonicity: We say h is *monotone* if $h(x_1, \ldots, x_m) \leq h(x'_1, \ldots, x'_m)$ whenever $x_i \leq x'_i$ for every i.

(A2) Decomposability: We say h is *decomposable* if, for any partition $\mathcal{P} = \{S_1, \ldots, S_t\}$ of $[m]$, there exists a function $g_{\mathcal{P}}$, and partial aggregation functions $f_{S_1}, \ldots, f_{S_t}$ such that $h(x_1, \ldots, x_m) = g_{\mathcal{P}}(f_{S_1}(\{x_j \mid j \in S_1\}), \ldots, f_{S_t}(\{x_j \mid j \in S_t\}))$.

For example, for $m = 6$, let $\mathcal{P} = \{\{1, 4, 6\}, \{2, 5\}, \{3\}\}$. Decomposability would mean that there are functions $f_{1,4,6}$, $f_{2,5}$, f_3, and $g_{\mathcal{P}}$ such that $h(x_1, x_2, x_3, x_4, x_5, x_6) = g_{\mathcal{P}}(f_{1,4,6}(x_1, x_4, x_6), f_{2,5}(x_2, x_5), f_3(x_3))$.

Many functions that occur in practice are monotone and decomposable. For example, if $h = \min$, max, or sum, the decomposition is easy — take $h = g = f$. Even complicated-looking functions such as $x_3 \log(x_1 + x_2)$ often have an easy decomposition.

[1]For notational simplicity, from here on, we will drop the braces when the subscripts are explicit sets. For example, we will use f_i instead of $f_{\{i\}}$ and $f_{i,j}$ instead of $f_{\{i,j\}}$.

3.2 Instance optimality

We are interested in *instance optimal* algorithms for top-k retrieval. Intuitively, instance optimality implies the optimality of algorithms on *every* input, not just on a worst case input. Formally, let $\mathbf{D}$ be a class of databases and $\mathbf{A}$ be the class of algorithms that correctly return the top k answers from $\mathbf{D}$ for every query. We say that an algorithm $\mathcal{B} \in \mathbf{A}$ is *instance optimal* over $\mathbf{D}$ and $\mathbf{A}$, if, for every $\mathcal{A} \in \mathbf{A}$ and every $\mathcal{D} \in \mathbf{D}$, $\text{cost}(\mathcal{B}) = O(\text{cost}(\mathcal{A}))$, where cost refers to the total amount of resources consumed by the algorithm. In our case, we will be making a distinction between *sequential* accesses with cost c_S per access, and *randomized* accesses with cost c_R per access. In typical scenarios, $c_R \gg c_S$.

3.3 Common subproblem

A common problem in the design of the early-termination condition for top-k algorithms, and in particular, TA and NRA, is to obtain a best and a worst case bound on the aggregated score for each element we have already seen and any element not yet seen.

Suppose that we can upper bound the score of each parameter i by $\underline{x}_i$, i.e., for every element X with scores $(x_1, \ldots, x_m)$, we know that $x_i \leq \underline{x}_i$. Then, by the monotonicity of the aggregation function, $h(x_1, \ldots, x_m) \leq h(\underline{x}_1, \ldots, \underline{x}_m)$. Indeed, this fact is crucial to proving the termination condition for TA and NRA [9]. Suppose now, that we know extra information on the aggregated score of some of the elements. How does this change our bounds?

As a running example, consider a case where $m = 3$ and the aggregation function h is the sum of all elements, i.e., $h(x_1, x_2, x_3) = x_1 + x_2 + x_3$. If all we know is $\underline{x}_1, \underline{x}_2, \underline{x}_3$ then an easy upper bound is $h(x_1 + x_2 + x_3) \leq \underline{x}_1 + \underline{x}_2 + \underline{x}_3$. Suppose, in addition, we know that $x_1 + x_2 \leq \underline{x}_{1,2}$, where $\underline{x}_{1,2}$ is an upper bound on the combination score of parameters 1 and 2. Then, we can obtain another upper bound: $h(x_1 + x_2 + x_3) \leq \underline{x}_{1,2} + \underline{x}_3$. Suppose that we also know the values of $\underline{x}_{2,3}$ and $\underline{x}_{1,3}$, then we can obtain a different upper bound $h(x_1, x_2, x_3) \leq \frac{\underline{x}_{1,2} + \underline{x}_{2,3} + \underline{x}_{1,3}}{2}$.

A quick enumeration leads to five possible upper bounds and we are interested in the smallest of them: $h(x_1, x_2, x_3) \leq \min\{\underline{x}_1 + \underline{x}_2 + \underline{x}_3, \underline{x}_{1,2} + \underline{x}_3, \underline{x}_{1,3} + \underline{x}_2, \underline{x}_{2,3} + \underline{x}_1, (x_{1,2} + x_{1,3} + x_{2,3})/2\}$. Some of the conditions could be redundant and finding the best possible upper bound calls for a principled approach. The way to solve this constrained maximization problem is by a linear program (LP). The above example can be formulated as a simple LP:

$$\max h(x_1, x_2, x_3) \text{ s.t. } x_i \leq \underline{x}_i, \forall i \text{ and } x_i + x_j \leq \underline{x}_{i,j}, \forall i, j. \quad (1)$$

More generally, we have the following setting. Let $\mathcal{S}$ be a family of subsets of $[m]$; we assume $\mathcal{S}$ contains all the singletons. Let ℓ be the maximum number of individual lists subsumed by a combination list, i.e., $\ell = \max_{S \in \mathcal{S}} |S|$. For each $S \in \mathcal{S}$, let $\underline{x}_S$ be an upper bound on the combination score of the parameters in S. Now, given a monotone, decomposable aggregation function h as above, we can express the best upper bound on h as a mathematical program:

$$\max h(x_1, \ldots, x_m) \text{ s.t. } f_S(\{x_j : j \in S\}) \leq \underline{x}_S, \forall S \in \mathcal{S}. \quad (2)$$

For an arbitrary h, this could be a complicated optimization. If however f_S, g, and h are convex, then the above mathematical program can be solved in polynomial time via convex programming.

For the rest of the paper we focus on the following important special case.

(A3) h is the *addition* function, i.e., $h(x_1, \ldots, x_m) = \sum_{i=1}^{m} x_i$.

Addition is a natural aggregation function that is of interest in IR and web search contexts: The relevance of a document to a multi-term query is the sum of the relevance of the document to each of the terms in the query.

3.3.1 Combinatorial algorithms for addition

For addition, notice that the mathematical program in (2) reduces to a simple LP.

$$\max \sum_{i=1}^{m} x_i \quad \text{s.t.} \quad \sum_{j \in S} x_j \leq \underline{x}_S, \forall S \in \mathcal{S}. \tag{3}$$

While this LP can be solved in polynomial time, it is not efficient in practice. The goal of this section to solve or approximate (3) using combinatorial methods. We first consider the important case of $\ell \leq 2$, i.e., we have pairwise combination lists. Next, we consider the general case and give an approximation algorithm.

Exact algorithm for $\ell \leq 2$. In the case when each combined list has at most two elements, i.e., $\ell \leq 2$, we give an exact combinatorial algorithm that is based on matching.

THEOREM 1. *If $\ell \leq 2$, then the LP in (3) can be solved by finding a minimum cost perfect matching on a particular graph on $2m$ nodes and $4|\mathcal{S}| + m$ edges.*

The proof is given by the following reduction. Let $E = \mathcal{S} \cap [m]^2$. We rewrite the LP in (3) as $\max \sum_{i=1}^{m} x_i$ s.t. $x_i \leq \underline{x}_i$ and $x_i + x_j \leq \underline{x}_{i,j}, \forall (i,j) \in E$. The dual of this LP is a covering LP:

$$\min \sum_{(i,j) \in E} y_{i,j} \underline{x}_{i,j} + \sum_{i=1}^{m} y_i \underline{x}_i \quad \text{s.t.} \quad y_i + \sum_j y_{i,j} \geq 1, \forall i. \tag{4}$$

Consider the following interpretation of the dual LP in (4). Let $G = (V, E)$ be a weighted graph on m nodes, and let $V = \{v_1, \ldots, v_m\}$. For each $(i,j) \in E$, let the weight of the (v_i, v_j) edge be $\underline{x}_{i,j}$, and the weight of node v_i be $\underline{x}_i$. Then the LP in (4) solves a fractional edge cover problem — we are to (fractionally) select edges and nodes so that the total weight emanating from any node is at least 1. We have the following characterization.

LEMMA 2. *The LP in (4) is half-integral, i.e., in the optimum, each $y_{i,j} \in \{0, \frac{1}{2}, 1\}$ and each $y_i \in \{0, 1\}$. Furthermore, the edges (i,j) with $y_{i,j} = \frac{1}{2}$ form a union of edge-disjoint cycles.*

PROOF. The current LP allows for selecting edges $y_{i,j}$ and nodes y_i to satisfy the covering constraints. We begin by transforming the graph so that only edge constraints are necessary. Let $H = (W, F)$, where W has two special nodes, $W = V \cup \{s, s'\}$. For each $v \in V$ we add an edge (v, s) with $\underline{x}_{v,s} = \underline{x}_v$. We add one extra edge (s, s') with $\underline{x}_{s,s'} = 0$. It is easy to see that there is a one-to-one mapping between solutions in graph H (for which $y_i = 0$), and solutions in G with (potentially) non-zero node constraints.

It is well known [20, p. 533] that the optimal solution to the problem on H, i.e., the edge cover problem, is half-integral with the half-integral edges composing edge-disjoint cycles. (While the half-integrality is explicit in [20, p. 533], the proof yields the cycle condition as well.) Therefore, each $y_{i,j}, y_i \in \{0, \frac{1}{2}, 1\}$. We now show that in fact each $y_i \in \{0, 1\}$. For any edge cover solution on H, if some edge (v_1, s) was selected with weight $\frac{1}{2}$, then some other edge (v_k, s) must have been selected with weight $\frac{1}{2}$ as well; this follows since half-integral edges lie on edge-disjoint cycles. Now, let $C = ((s, v_1), (v_1, v_2), \ldots, (v_k, s))$ be the cycle including these two edges. By [20], for every $e \in C$, $y_e = \frac{1}{2}$. Suppose k is even (the k odd case is similar). Let $C^{\text{odd}} = ((s, v_1), (v_2, v_3), (v_k, s))$ and $C^{\text{even}} = ((v_1, v_2), (v_3, v_4), \ldots, (v_{k-1}, v_k))$. Consider two solutions

$$y_e^{\text{odd}} = \begin{cases} 1 & \text{if } e \in C^{\text{odd}} \\ 0 & \text{otherwise,} \end{cases} \quad \text{and} \quad y_e^{\text{even}} = \begin{cases} 1 & \text{if } e \in C^{\text{even}} \\ 0 & \text{otherwise.} \end{cases}$$

Each of these solutions is feasible — since s' is of degree 1, the edge (s, s') is always selected, and thus s need not be covered by C. And since $y = \frac{1}{2} y^{\text{odd}} + \frac{1}{2} y^{\text{even}}$, one of the two solutions is cheaper. Therefore, an optimum solution to (4) will have each node weight $y_i \in \{0, 1\}$. □

We now transform G to obtain a combinatorial method for solving (4). Let G' be a graph obtained from G by adding a disjoint copy $\tilde{G} = (\tilde{V}, \tilde{E})$, and adding for each edge $(v_i, v_j) \in E$ cross edges $(v_i, \tilde{v}_j)$ and $(\tilde{v}_i, v_j)$ with weight $\underline{x}_{i,j}$. Furthermore, we add edges $(v_i, \tilde{v}_i)$ with weight $2\underline{x}_i$. Thus G' has $2m$ nodes and $4|S| + m$ edges in total.

LEMMA 3. *The cost of the minimum cost perfect matching in G' is twice the cost of the optimal fractional edge cover in G.*

PROOF. Consider a perfect matching M in G'; it is easy to check that a perfect matching is guaranteed to exist. We construct a solution to (4) of half the cost. Let $y_i = 1$ if $(v_i, \tilde{v}_i) \in M$ and 0 otherwise. Denote by $I_M(e)$ the indicator variable for the matching, i.e., $I_M(e) = 1$ iff $e \in M$. Then let

$$y_{i,j} = \frac{1}{2} I_M(v_i, v_j) + \frac{1}{2} I_M(v_i, \tilde{v}_j) + \frac{1}{2} I_M(\tilde{v}_i, v_j) + \frac{1}{2} I_M(\tilde{v}_i, \tilde{v}_j).$$

It is easy to check that the solution y on G defined above is exactly half the cost of the matching. Now we focus on the constraints. To see that $y_i + \sum_j y_{i,j} \geq 1$, observe that y_i is either 1 or 0. Since M is a perfect matching, if $y_i = 0$, there exist $u, \tilde{u}$ such that $(v_i, u) \in M$ and $(\tilde{v}_i, \tilde{u}) \in M$. Thus the total edge weight surrounding v_i is at least 1, thereby satisfying the constraint in (4).

Conversely, we construct a matching M from a solution to (4). If $y_i = 1$, add the edge $(v_i, \tilde{v}_i)$ to M. If $y_{i,j} = 1$, add (v_i, v_j) and $(\tilde{v}_i, \tilde{v}_j)$ to the matching M. We are left with edges $F \subseteq E$ such that for all $e \in F$, $y_e = \frac{1}{2}$. From Lemma 2, we know that the edges in F form a union of edge-disjoint cycles. Orient each cycle so that it becomes directed. For each directed edge (v_i, v_j) in the cycle, add the edge $(v_i, \tilde{v}_j)$ to the matching M. Again, it is easy to check that the total cost of M is exactly twice the cost of the fractional edge cover. To ensure that it is a matching, we appeal to Lemma 2. The edges corresponding to integral y's never meet at a node. The edges (v_i, v_j) corresponding to $y_{i,j} = \frac{1}{2}$ form edge-disjoint cycles, and in G' each node is covered exactly once: v_i by the tail of the directed edges, and $\tilde{v}_i$ by the head of the edges. □

This completes the proof of Theorem 1.

Approximation algorithm. We now show a simple approximation to (3). First, we state the following notion. The lower bound values $\underline{x}_S$ given in (3) are said to be in *reduced form* if they satisfy the following two conditions: (B1) $\forall i \in [m]$, $\underline{x}_i \leq \min_{S \in \mathcal{S}, S \ni i} \underline{x}_S$, and (B2) $\forall S \in \mathcal{S}$, $\underline{x}_S \leq \sum_{j \in S} \underline{x}_j$. Essentially, by reduced form we mean that it is not possible to use some of the lower bound values to obtain a better lower bound on some other. Now, let ψ be the solution to (3) and recall that $\ell = \max_{S \in \mathcal{S}} |S|$.

THEOREM 4. *Let $x'_i = \underline{x}_i / \ell$ and let $\psi' = \sum_{i=1}^{m} x'_i$. If (B1) and (B2) hold, then $\ell \cdot \psi' \geq \psi \geq \psi'$.*

PROOF. Consider $x''_i = \ell \cdot x'_i = \underline{x}_i$. We claim that none of the constraints in (3) is satisfied by x''_i's with a strict inequality. To see this, suppose we have $\sum_{j \in S} x''_j \lneq \underline{x}_S$. Then, by (B2),

$$\sum_{j \in S} \underline{x}_j = \sum_{j \in S} x''_j \lneq \underline{x}_S \leq \sum_{j \in S} \underline{x}_j,$$

which is a contradiction. By duality,

$$\psi \leq \sum_{i=1}^{m} x''_i = \ell \cdot \sum_{i=1}^{m} x'_i = \ell \cdot \psi'.$$

We now show that the x'_i's satisfy every constraint in (3), i.e.,

$$\sum_{j \in S} x'_j = \frac{1}{\ell} \cdot \sum_{j \in S} \underline{x}_j \leq \frac{1}{\ell} \sum_{j \in S} \underline{x}_S = \underline{x}_S \frac{|S|}{\ell} \leq \underline{x}_S,$$

where the first inequality follows from (B1) and the second follows from the definition of ℓ. Hence, the x'_i's are a feasible solution to (3), yielding $\sum_{i=1}^{m} x'_i = \psi' \leq \psi$. □

Thus, we obtain an easy ℓ-approximation algorithm for the LP given in (3). (It is important to note that this approximation factor does not translate to any approximation with respect to the efficiency of TA and NRA algorithms.) As an aside, we note that this approximation factor is tight for this algorithm. Consider a setting where $\ell = 2$, $\underline{x}_i = 1$ for all i, and $\underline{x}_{i,j} = 1$ for all i and j. Using the setting in Theorem 4, we get $\psi' = m$, whereas the optimal solution to (3) is attained by setting $x_i = 1/2$, which results in a bound of $\psi = m/2$.

3.4 Threshold algorithm

The threshold algorithm (TA) introduced by Fagin et al. [9] combines sequential access to each of the lists with random accesses to compute each element's total aggregation score. The addition of combination lists results in an interesting question for TA — should we do sequential access on all of the lists, or does the traversal of the combination list make the traversal of the individual lists a wasted effort? For example, given that we have a list $L_{1,2}$ should we traverse lists L_1 and L_2? We show that to maintain instance optimality TA *must* traverse *all* of the available lists.

We begin by recalling the TA algorithm and adapting it to the situation with the additional combination lists. We then prove matching lower and upper bounds on the instance optimality ratio.

3.4.1 Algorithm

The TA algorithm begins by doing sequential accesses in parallel into each of the available sorted lists (singleton as well as combination lists). For each object X that has been seen, the algorithm completes the fields of X by doing random accesses into the other lists and computes the score $h(X)$ using the full information.

At the same time, let $\underline{x}_S$ be the score of the last object seen in list L_S. We define the threshold value ψ as the solution to the mathematical program presented in (2). Note that ψ is non-increasing by monotonicity. As soon as we find at least k elements with value at least ψ we halt and output the objects with the highest score.

For (3), we first show the following.

LEMMA 5. *The lower bound values obtained at each step of TA are in reduced form.*

PROOF. Since TA scans the list in parallel, from the descending order and non-negativity of scores, the r-th largest score in L_T, i.e., $\underline{x}_T$, is clearly at most the r-th largest score in any list L_S (i.e., $\underline{x}_S$), for $T \supseteq S$. This yields (B1). Similarly, the r-th largest entry in the list L_S (i.e., $\underline{x}_S$) is at most the sum of the r-th largest in the lists corresponding to any disjoint partition of S. This yields (B2). □

Using Theorem 1 for $\ell = 2$, and Theorem 4 otherwise (as implied by Lemma 5), we obtain a combinatorial algorithm to determine the stopping condition given by (3).

3.4.2 Instance optimality

Let $s = |\mathcal{S}|$ be the total number of lists. We show that TA is instance optimal with optimality ratio sm. (Here, we consider c_R and c_S to be constants.) We then show that this is indeed the best ratio one can obtain.

THEOREM 6. *Let h be an arbitrary monotone aggregation function. Let $\mathbf{D}$ be the class of all databases and $\mathbf{A}$ be the class of algorithms that correctly identify the top k answers for h for every database and do not make wild guesses[2]. Then TA is instance optimal over $\mathbf{A}$ and $\mathbf{D}$ with optimality ratio sm.*

The proof of this theorem parallels directly the proof of the original TA algorithm presented in [9]. We omit it here.

THEOREM 7. *Let h be an arbitrary monotone aggregation function with m arguments. Let $\mathbf{D}$ be the class of all databases. Let $\mathbf{A}$ be the class of all algorithms that correctly find the top k answers for h for every database and that do no make wild guesses. There is no deterministic algorithm that is instance optimal over $\mathbf{A}$ and $\mathbf{D}$, with optimality ratio less than $s + sm'c_R/c_S$, where s is the total number of lists, and m' the minimum number of lists that together allow us to compute the element score.*

PROOF. We proceed by proving the $k = 1$ case, the generalization to $k > 1$ is simple. Let us fix the following parameters: (1) d, k_1, k_1 are integers, (2) $\psi = (ds - 1)c_S + (ds - 1)m'c_R$, and (3) $k_2 > k_1 > \max(d, \psi/c_S)$. Our argument follows closely the lower bound presented in [9] in the case without combination lists.

We restrict our attention to a special family $\mathbf{D}'$ of databases of the following form. In every individual list the top k_2 scores are 1, and the remaining scores are 0. In every combination list on q elements the top k_2 scores are q, which are then followed by a number of scores of $q - 1$, $q - 2$, etc. No object appears in the top k_1 of more than one list and there is one object X which has score 1 in all of the individual lists, and s' in all of lists on s' elements. This object is in the top d of exactly one list. Except for X every object has a score of 0 in at least one of the individual lists, and therefore has score $s' - 1$ in some of the s element lists. Finally, we claim that given sufficiently many objects, we can pick k_1 and k_2 to satisfy the conditions.

As in [9], we say that an object is *high* in list i if it appears in the top d of the list, and is generally high if it is high in one of the lists. Let $\mathcal{A} \in \mathbf{A}$ be an arbitrary deterministic algorithm. Suppose $\mathcal{A}$ sees $(sd - 2)$ high objects in total, in other words, there are two high objects that it does not see. Then $\mathcal{A}$ cannot correctly decide which one of the two remaining maximizes h, and thus $\mathcal{A}$ must see at least $sd - 1$ objects and have cost at least $(sd - 1)c_S$.

In counting the number of random accesses, there are two cases. If $\mathcal{A}$ ever sees some object in list j that is high in list $i \neq j$, then it must have scanned past k_1 in list j and thus its cost is at least ψ.

Otherwise, we say that an object Y is fully randomly accessed if after seeing it sequentially in list i, $\mathcal{A}$ figures out the total score for Y. The cheapest way of doing this is observing the position of Y in the m' lists that together cover all of the parameters. We can always reveal the values adversarially in such a way that all but the last of such accesses will produce the maximum score. Therefore, before the last access the object always has a possibility of being X. Again, as long as there are two high objects X_1 and X_2 that have not been fully randomly accessed, the algorithm cannot determine whether $h(X_1) > h(X_2)$ or vice versa. Therefore, it needs at least $(ds - 1)(m')$ random accesses with the total cost ψ. Taking d sufficiently large, the ratio $\frac{\psi}{(sd-1)c_S}$ can be made arbitrarily close to $s + sm'c_R/c_S$ as required. □

COROLLARY 8. *If TA does not explore all of the possible lists, then it is not instance optimal.*

[2]An algorithm $\mathcal{A}$ makes no wild guesses if the first access to every element is a sequential access.

PROOF. Consider the construction in Theorem 7. We can place the top object X in position k_2 in all of the lists where it is not a high element. If the list in which the top object X appears at the first d elements is ignored, then TA must make at least k_2 accesses with a cost of $k_2 c_S \gg \psi$. $\square$

3.5 No random access algorithm

Unlike the TA algorithm, the NRA algorithm does not make any random accesses throughout the list. As such, it cannot know the full score for an element X until it has accessed all of its values through sequential list access. In the case with no combinations, this happens when X is accessed in all of the lists. However, for every item X, the algorithm can maintain a range of values that X can take on. Since we have assumed that each individual score of X lies between 0 and 1, and the target function h is monotone, we can always make such a computation.

Suppose, for example, that $m = 6$ and we have learned the values of x_1, x_3, and x_6 for X. Then we can say that $h(x_1, 0, x_3, 0, 0, x_6) \leq h(x_1, \ldots, x_6) \leq h(x_1, 1, x_3, 1, 1, x_6)$.

While we cannot improve the lower bound, we can use the fact that we are reading the lists in decreasing order to improve the upper bound. Denote again by $\underline{x}_S$ the last value read in list L_S. Then we can express the upper bound for $h(X)$ similar to (2), except with the additional constraints reflecting our current knowledge of X. Let T be the set of variables that have been revealed.

$$\max h(y_1, \ldots, y_m) \text{ s.t. } y_i = x_i, \forall i \in T \text{ and } y_i \leq \underline{x}_i, \forall i \notin T \text{ and } f_S(\{y_j : j \in S\}) \leq \underline{x}_S, \forall S \in \mathcal{S}, S \not\subseteq T. \quad (5)$$

Using (5) as a black box, we describe the new NRA algorithm.

(1) Do sequential access to each of the lists $L_S, S \in \mathcal{S}$. As before maintain the bottom values $\underline{x}_S$ encountered in list S.

(2) For each element X, compute the best and worst possible scores for X, $B(X)$ and $W(X)$. We can compute $B(X)$ using (5) and $W(X)$ by substituting the value 0 for all elements that we have not yet seen.

(3) Let M be the k-th largest W value, with ties broken in favor of higher B values.

(4) Halt when at least k objects have been seen and for every object X that is not in the top k, $B(X) > M$.

Similar to the TA case, for (3), we obtain combinatorial algorithms to determine the stopping condition given by (5) using Theorem 1 for $\ell = 2$, and Theorem 4 otherwise.

3.5.1 Instance optimality

The instance optimality of NRA relies on the algorithm having the optimal upper bound for the range for each list. Equivalently, there must always exist an assignment of the remaining variables such that (i) the upper bound on the range is achieved and (ii) the assignment is consistent with all of the available information.

This information is easy to obtain in the case when we have only lists for individual attributes. The upper bound is obtained by setting $y_i = x_i$ for $i \in T$ and $y_i = \underline{x}_i$ for $i \notin T$. In the case where combined information is available, the situation becomes more intricate. For example, if we know that $x_1 = a$ and $\underline{x}_1 + \underline{x}_2 \leq b$, then we can quickly conclude that $\underline{x}_2 \leq b - a$. This information can potentially be used further to obtain a bound on $\underline{x}_3$, and so on.

The mathematical program presented in (5) encapsulates all of this information, and gives the highest possible feasible value for $h(X)$. The objective function is the upper bound on the range of X and the assignment to the individual variables y_i realizes this bound and is consistent with all of the known information.

Using the observation above, the following theorem follows after an easy adaptation of the proofs in Fagin et al. [9].

THEOREM 9. *Let h be a monotone aggregation function and let $\mathbf{D}$ be the class of all databases and $\mathbf{A}$ be those algorithms that correctly identify the top k answers for h for every database and make no random accesses. Then NRA is instance optimal over $\mathbf{A}$ and $\mathbf{D}$ and no deterministic algorithm has a lower optimality ratio.*

4. EMPIRICAL ANALYSIS

In this section we evaluate our early-termination query-processing algorithms in the presence of pre-aggregated lists. We focus our attention on pairwise combinations, or *intersections*, i.e., for a given pair of terms, the pre-aggregated list corresponding to the pair comprises only those documents that have *both* of the terms. (In other words, the score of a document not containing both terms is set to $-\infty$.) We present our evaluation in two parts. In the first part we drill down into the special case of queries with three terms and perform exhaustive experiments with different numbers and sets of intersection lists being available. These experiments are run on the TREC GOV2 data set and help quantify the benefits of having more intersection lists. In the second part we present an evaluation of our approach on a real-world web search index and query load.

4.1 Evaluation methodology

4.1.1 Datasets

GOV2. This is a standard data set used in the annual TREC competition by IR researchers. It consists of 25.2M web pages from the `gov` top-level domain crawled in early 2004. The pages consist of HTML and text files, plus the extracted text of any pdf, word, and postscript documents truncated to 256 KB. In addition, a set of 100,000 queries is provided. These queries are, however, not known to constitute a typical query load for a general web search engine. Therefore, we will use an index constructed on this data set to primarily provide an initial detailed analysis of our algorithms, and then add results from another data set to get an overall view.

Web search data. In order to show the benefit of running our algorithms under a real-world search engine query load, we use queries presented to the Yahoo! search engine. We use a query load of around 1B queries spanning multiple days to decide what intersections to pre-compute; the particular method used for this is described later in Section 4.3. From the query load of subsequent days we randomly select a set of 10,000 queries that we use to evaluate the performance of our algorithms. The search index used in these experiments was constructed to simulate query execution on a single machine in a distributed search system. We indexed a set of around 16M randomly selected web pages.

In order to make our experiments conform closely to real-world settings, we tried to simulate the effects of various types of caching performed by commercial search engines. One of the most important ones is *results caching*, where the top-k results of a frequently occurring query are cached in order to avoid processing it repeatedly. Results caching affords enormous advantages to commercial search engines and reduces the number of repeated queries seen by the query processing system. This has a significant effect on the distribution of queries that need to actually be executed; Figure 1 shows that the average length of unique queries is larger than the average length of all queries. We approximate the effect of result caching by eliminating duplicate queries from the query traces used in our evaluation. (This is known to have a similar effect as common result caching policies.)

4.1.2 Measures

Following previous work on query processing systems we will report our results using two intuitive measures of performance.

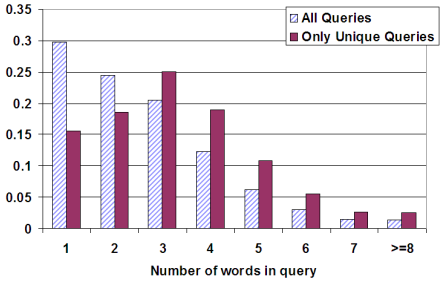


Figure 1: Query word distribution.

Number of sequential accesses (NSEQ). Recall that our algorithms involve descending a set of posting lists that each order web pages by the score of the corresponding term in the web page's ranking (scored by a relevance function such as the BM25 function). The NSEQ measure captures the number of times the cursor is moved forward in the posting lists and provides a straightforward notion of efficiency for algorithms. Clearly smaller values of NSEQ indicate better performance.

Number of random accesses (NRND). This measure only applies to the TA algorithm and is the number of random lookups into lists. Once again smaller values of NRND means better performance. Each sequential accesses in TA can lead in one or more random lookups, depending on whether the appropriate intersections are available or not. Also, we do not count a lookup if the object we look for already appeared earlier in a sorted access in another list. Because of these reasons, there is no simple relationship between NSEQ and NRND, and we report both for the case of TA. In general, the cost of random accesses far outweighs the cost of sequential accesses; previous work has used factors of 100 to 1000 [19].

4.1.3 Baseline algorithms

Since we are combining two different paradigms — pre-aggregating posting list intersections and early termination — for speeding up query evaluation, we evaluate our approaches against three baseline algorithms described below.

No intersections (NOINT). This baseline assumes that the pre-aggregation of posting list intersections is unavailable, and that early-termination algorithms like TA and NRA are run on the original posting lists for the query terms. Comparing against this approach gives an idea of how well our algorithms exploit the intersection lists for early termination. The efficiency of this baseline is evaluated in terms of the above two measures.

No early termination (NOET). In this baseline we perform full query evaluation but assume that all possible intersections are available. Note that this baseline does not scan an original list if it is covered by an intersections — this only strengthens the baseline versus our algorithms. Our TA and NRA algorithms will scan both the original and the intersection lists, since without scanning all lists they would not remain instance optimal (Corollary 8). Also note that the intersections of posting lists are typically much smaller in size than the original lists. Hence, this baseline helps us isolate the inherent advantage to query processing when intersections of posting lists are available. The way we measure the performance of this baseline depends on the algorithm with which it is compared. When the baseline is compared with NRA, the NSEQ measure is computed as the sum of the sizes of all lists that together cover all the query terms. When comparing against TA, we assume that structures for random lookup into individual posting lists are available to the baseline. In this case NSEQ is the size of the shortest posting list (including intersections) and NRND is NSEQ times the number of query terms not covered by the shortest posting list. (This is meant to account for the ability of state-of-the-art IR query processors to perform forward skips from the shortest into the longer lists for improved efficiency.)

No intersections, no early termination (NOINTNOET). In this baseline we apply full query evaluation in the absence of intersections of posting lists. The performance of this baseline is scored in the same way as for the NOET baseline.

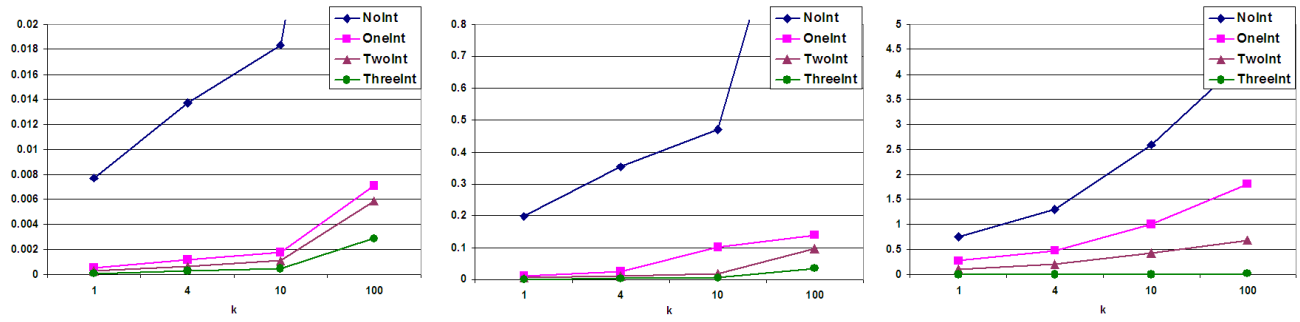
4.2 Detailed analysis of three-word queries

In this section we dive deep into analyzing three-word queries on an inverted index created with the TREC GOV2 data set. Given the published, clean nature of the data set, we hope that the results of our experiments can be used by other researchers in future work. All results reported in this section are obtained by averaging the results of executing 100 three-word queries in various scenarios. Note that for the cases of one and two available intersections, there are three possible cases of which intersections are available. For instance, for a query with terms {A,B,C} and one available intersection, this available intersection could be either AB, AC, or BC, and performance might significantly vary between these cases. For this reason, we report results by executing all three cases and averaging the numbers.

For presentation of the results, we plot the relative performance of the algorithms with respect to the baseline NOET. The rationale for this is twofold. First, we would like to understand and quantify the extent to which early termination helps even in the presence of intersection lists that are potentially very small. Second, the absolute performance of NOET is often so far off the scale that including it as a curve along with the other algorithms would make the plots less readable.

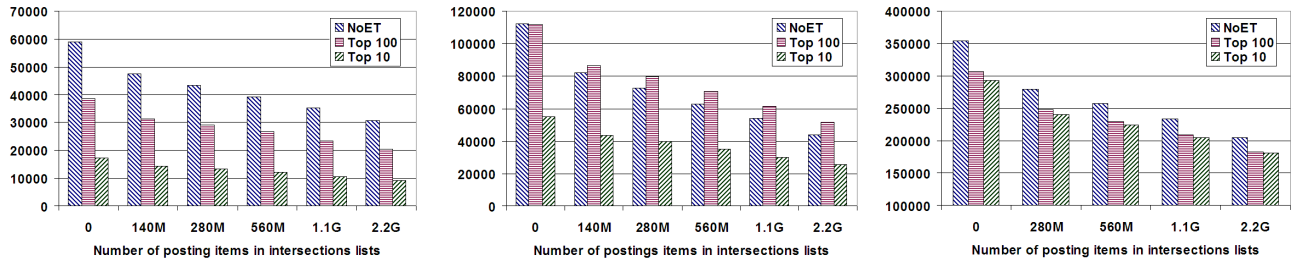
Figures 2(a) and 2(b) show the performance of TA on three-word queries. The first thing to notice is that the baseline NOINT is orders of magnitude worse than having even a single intersection list for both NSEQ and NRND for TA. Furthermore, additional intersection lists offer a pronounced improvement; even going from two intersections to three intersections results in a non-negligible speedup. Second, observe the y-axis of the plots: Since we are comparing the measures relative to NOET, even with three intersection lists, early termination has tremendous benefits with respect to the NSEQ and NRND measures. Thus, having intersections only has limited benefits, and adding early termination to the intersections results in significant additional improvements. The third aspect to note is that as more intersections are available, even computation of top-k results for larger values of k is highly efficient, with a graceful degradation as k increases.

Figure 2(c) shows a similar plot for the performance of the more realistic NRA algorithm on three-word queries. In this case, we only have one measure, for sequential accesses (NSEQ). While the overall trends follow the TA experiment, there are some crucial differences. First, the gap between using no intersection lists and using one intersection list is less pronounced than in the TA case. Second, the gap between, say, using one intersection list and two intersection lists is more pronounced. In other words, the benefits of more intersections is more significant. The third important point to notice is the y-axis: For large values of k, the baseline NOET actually performs better than early-termination algorithms with intersection. (Recall that NOET scans only the intersection lists, whereas NRA scans both the original and intersection lists.) In other words, for NRA using only intersections is more beneficial than doing early termination with no intersections (or one intersection, for $k = 100$), while for TA the opposite is true. Both TA and NRA, however, benefit from combining intersections and early termination, with TA achieving more significant improvements.



(a) Algorithm: TA, Measure: NSEQ (b) Algorithm: TA, Measure: NRND (c) Algorithm: NRA, Measure: NSEQ

Figure 2: Performance of TA and NRA on three-word queries on the TREC GOV2 data set. The values on the y-axis are the ratio of the NSEQ and NRND values of the early-termination algorithms and the corresponding values for the NOET baseline.



(a) Algorithm: TA, Measure: NSEQ (b) Algorithm: TA, Measure: NRND (c) Algorithm: NRA, Measure: NSEQ

Figure 3: Performance of our early-termination approaches compared with full query execution (NOET). The x-axis represents different amounts of precomputed intersection list data. The total index consisted of about 2.2B postings, and thus the rightmost set of bars means an increase in index size by about a factor of 2. We note here that NOET is also subjected to the limit on available intersections, while in Figure 2 it had access to all possible intersections.

4.3 Results on a web search index

In this section we evaluate the performance of our early-termination algorithms on the web search data set described in Section 4.1.

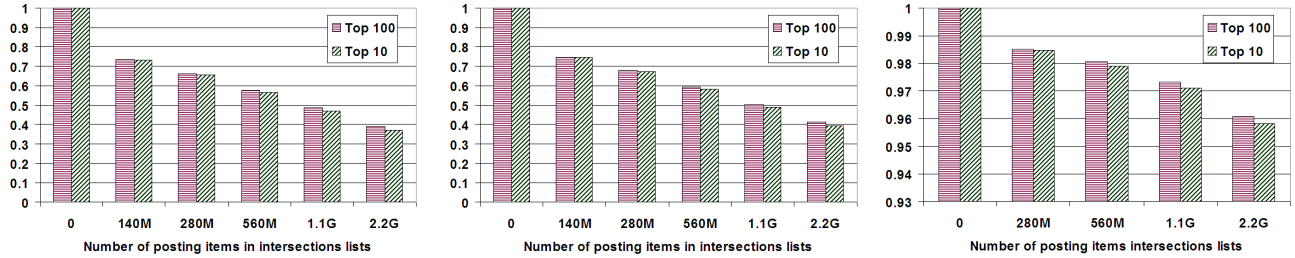
Intersection selection methodology. In order to select a good set of posting list intersections that are precomputed, we used a query load of 1B queries seen over multiple days at the Yahoo! search engine. The algorithm we used to select the posting lists to intersect and cache is a slightly modified version of the offline algorithm by Long and Suel [14]. The original algorithm is used to select projections to maintain in a cache and hence considers the benefit given by replacing a posting list by its projection. In our current application we replace this measure by the benefit of replacing two posting lists by their intersection. In other respects the algorithm is exactly the same. We want to isolate the improvement obtained by using the intersections with our approach and hence do not investigate sophisticated ways of selecting posting lists to intersect and store. This way we can factor out the benefits of a sophisticated intersection selection approach and just showcase the benefits of our approach to processing the intersections. Note that the above mentioned methodology can be applied to select both full intersections as well as fixed size prefixes of intersections.

Given the above algorithm to select intersections, the experimental setup is as follows. We fix the size of the cache and fill it with intersections chosen according to the above algorithm. We then run our versions of TA and NRA (both using matching-based upper bounds) to process a test set of 10,000 queries randomly sampled from the Yahoo! search engine query logs. While executing these queries, the generalized TA and NRA algorithms are allowed to take advantage of all intersection lists available in the cache.

Comparison with baselines. In Figures 3(a) and 3(b) we plot the performance of our TA algorithm when run with various numbers of intersection lists. The performance numbers are in terms of the raw NSEQ and NRND values averaged over the test query trace and are plotted against increasing sizes of the intersections cache. In all our plots the size of the intersections cache is expressed in terms of number of posting list elements. The three bars correspond to the NOET baseline, and TA for top-10 and top-100 documents. Note that the left-most bar in both plots correspond to running NOET in the absence of any intersection lists, which is the same as running the NOINTNOET baseline. A similar graph displaying the NSEQ values for our NRA algorithm is in Figure 3(c).

We can see from the plots that for top-10 query processing under all settings of sequential/random access and different amounts of available intersections, our algorithms significantly outperform the baseline. For example, when we run our TA algorithm with a cache size of around 560M postings, we perform an average of 80% fewer sequential accesses (NSEQ) and 70% fewer random accesses (NRND) than the NOINTNOET baseline. For the case of performing NRA with the same sized intersection cache the NSEQ measure is reduced by an average of 37%. Moreover, the performance of TA/NRA continues to improve as we increase the number of term intersections that we cache.

For the NOET baseline, the improvement over the NOINTNOET baseline for 560M postings is around 34% (NSEQ) and 45% (NRND) in the TA setting and 27% for NSEQ in the NRA setting. These numbers are significantly smaller than the corresponding numbers for our algorithms, highlighting that caching intersections is not enough by itself and that adding early termination gives significant additional benefits. For the NOINT baseline, performance improve-



(a) Algorithm: TA, Measure: NSEQ (b) Algorithm: TA, Measure: NRND (c) Algorithm: NRA, Measure: NSEQ

Figure 4: Performance of our algorithms when using matching-based upper-bound computation, relative to performance when using approximation algorithm-based upper-bound computation. That is, the values on y-axis of (a) are the ratio of NSEQ of TA using matching-based upper bounds and NSEQ of TA using the approximation algorithm.

ments from only performing early termination are 70% (NSEQ) and 50% (NRND) for TA and 18% (NSEQ) for NRA. Once again these numbers are lower than the improvements in performance seen by our approach.

These results are significant in practice because of two reasons. First, as we mentioned earlier, our approach to selecting the intersection lists is quite naive. Using better methods, we could not only increase the coverage of intersection lists, i.e., the fraction of queries that have access to at least one intersection list, but also choose those intersections that will yield the highest value in terms of performance. Second, in these experiments we retained the whole intersection list. In practice, a prefix of the intersection list should usually suffice for early termination purposes. Pruning frees up space to store even more intersections, and therefore can boost the performance even further. Later in this section we perform some initial naive experiments to show that caching prefixes of intersection lists helps us get further performance improvements.

The results for top-100 query evaluation are overall similar. For the NRA setting, our algorithm improves upon NOINTNOET by 35% (with 560M intersection postings), outperforming the other baseline methods. The baselines NOINT and NOET show improvements of only 14% and 27% respectively. However, for large values of k the benefits of early termination decrease, and the performance of TA can sometimes be worse than that of the NOET baseline. We observed the same behavior in top-100 query evaluation in our experiments with the GOV2 data set (Figure 2). As explained, since our algorithms need to perform sequential access on all available lists to remain instance optimal, sometimes the number of accesses can be larger than for full query evaluation. This usually happens when some lists consistently provide leads to web pages that do not make it into the top-k results. This raises a question that we investigate next.

Is it necessary to scan all posting lists? In Corollary 8, we showed that any algorithm that works with intersection lists has to scan all of the lists in order to remain instance optimal. However, does this worst-case result hold in practice? What if an algorithm tries to be clever and examines only the intersection lists (since they are smaller) plus those original lists that are not covered by an intersection? We compared the performance of this algorithm, called sub-TA, against our generalized TA algorithm on the same data set, with 560M postings of intersection data and computing top-10 results. It turns out that sub-TA is better than TA for 89% of the queries. In those cases, the average difference in NRND is 4,558. This validates the belief that in majority of cases, it is not necessary to examine all the original input lists. However, in the remaining 11% of the cases when TA performs far better than sub-TA; the average difference in NRND is 127,152. This shows that even though sub-TA might cut down random accesses on a majority of the cases, in the cases where it loses to TA, it loses very badly, and the average performance of sub-TA is worse than that of TA. Thus, the worst-case scenarios painted in Corollary 8 can actually occur in practice, and if all the original lists were not scanned a lot of accesses may be required to unearth a good result.

Performance of approximation based upper-bound computation. Our TA and NRA algorithms need to compute upper bounds on the score that can be achieved by any pages that have not yet been encountered in the posting lists. In Section 3, we presented an exact combinatorial algorithm based on minimum cost perfect matching, and a simple 2-approximation algorithm for (3). Using the approximation algorithm for upper bounds gives us a small gain in speed during early termination execution, but the gain is low since matching is typically computed on a small graph and only executed every few steps instead of each time. However, the overestimation of the upper bound by the approximation algorithm could hurt us by delaying termination of the algorithm, and thus lead to a larger number of sequential (and random) accesses. Here we perform experiments to verify if it is worth substituting the matching-based approach for the 2-approximation algorithm when $\ell = 2$.

In Figures 4(a)-4(c) we compare the performance of TA and NRA with matching-based and approximate upper-bound computations. The x-axis once again represents the amount of space allocated for intersections. The y-axis shows the ratio of the performance of the matching-based algorithm to the performance of the approximation. Recall that an approximation factor on the upper bounds does not translate into an approximation factor on the total performance. Indeed, when the number of postings (and hence intersection lists) cached is large, the approximation-based TA does almost three times more work than the matching-based TA.

Recall that the extra intersections improve the performance of retrieval algorithms in two ways. First, the intersection lists help the algorithm discover new elements (an element x may be far down in lists L_i and L_j, but near the top in list $L_{i,j}$). Second, the intersection lists improve the bounds on the unseen elements as computed by the mathematical program. A careful analysis of the approximation based approach shows that its improvement comes solely from discovering new elements, and that the upper bounds calculated do not even take into account the scores seen in the intersection lists. Thus Figures 4(a)-4(c) show us the extra power gained by computing tighter bounds. Overall, the improvement in performance is well worth the small amount of additional time needed to compute the exact solution to (3) once every 100 or more steps.

Performance when caching prefixes of intersections. As we mentioned earlier, we expect that the performance of our algorithms in real-world settings can be further improved if we cache prefixes as

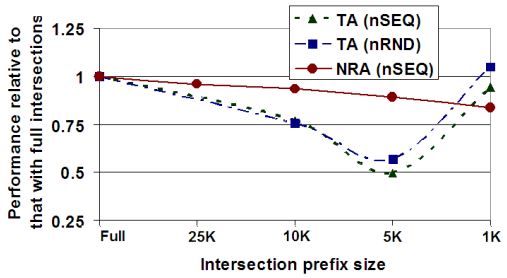


Figure 5: Performance of TA and NRA when prefixes of intersections are cached. The x-axis denotes the size of the prefixes. This experiment was performed with a cache size of 560M postings for top-10 results. Values on y-axis are the performance of our algorithms with prefixes of the indicated size relative to the performance with full intersections.

opposed to full intersections. Here, we perform preliminary experiments to support this assertion. As before, our objective is not to provide the best way to pick which intersections and what length of prefixes to cache, but to isolate the performance boosts afforded by our algorithms in the presence of prefixes. In the experiments, we fixed the size of the intersection cache to 560M postings. For each experiment, we fill the cache with prefixes of intersections of a fixed specified size. The scheme to pick intersections is the same as the one mentioned above, which relies on the the offline algorithm in [14]. Then, as before we execute our test set of 10,000 queries using our generalized TA and NRA algorithms using matching-based upper bounds.

Figure 5 plots the performance of our TA and NRA algorithms on the top-10 task. The x-axis denotes the different sizes of the prefixes, and the y-axis shows the ratio of the performance of TA/NRA when using these prefixes to the performance when using full intersections. As we can see, using only prefixes of a smaller length, we get lower values of NSEQ and NRND indicating better performance. When caching prefixes of size 5K, the TA algorithm needs on average half as many accesses (both NSEQ and NRND) to find the top-10, while the performance of NRA improves by only 10%. The intuition behind this result is that early-termination algorithms typically only access a small prefix of cached intersection lists. Caching smaller prefixes then means that we can use the remaining space to pre-aggregate more intersections, and thus have a better chance of having one or more useful intersections available for a query. If, however, the prefix size we select is too small, then performance decreases because such prefixes are not long enough to help algorithms terminate faster. This phenomenon is seen in Figure 5 at prefix size 1K where the performance of TA becomes worse than that with full intersections. This experiment indicates that a more careful selection of intersections and prefix sizes could result in significant additional improvements.

5. CONCLUSIONS AND FUTURE WORK

In this paper we obtained generalizations of the TA and NRA algorithms to the case when some pre-aggregated intersections of postings lists are available in addition to the original lists. Our generalization is based on computing appropriate upper and lower bounds implied by the available information, using a mathematical program. For the special case of the addition aggregation function, we obtain a matching-based algorithm for pairwise intersections, and a linear program (that can be approximated) for intersections over larger numbers of lists.

We conducted experiments on indices built using the TREC GOV2 data set and a few million randomly selected web pages from a search engine. Through an in-depth analysis of our algorithms on real-world query traces, we quantified the gains of both early termination and intersection lists. We showed that combining both techniques results in significant performance gains over each individual technique.

In terms of future work, an interesting open problem is to obtain a combinatorial algorithm for determining the stopping condition for the $\ell > 2$ case. We would also like to characterize cases when it is acceptable in practice to not scan all the intersections and original lists. Finally, a good solution to the problem of which intersections and which sizes of prefixes to cache might give significant additional improvements.

Acknowledgments. We thank Prabhakar Raghavan, Eva Tardos, and David Williamson for useful pointers.

6. REFERENCES

[1] V. N. Anh, O. de Kretser, and A. Moffat. Vector-space ranking with effective early termination. In *Proc. 24th SIGIR*, pages 35–42, 2001.
[2] V. N. Anh and A. Moffat. Compressed inverted files with reduced decoding overheads. In *Proc. 21st SIGIR*, pages 290–297, 1998.
[3] V. N. Anh and A. Moffat. Pruned query evaluation using pre-computed impact scores. In *Proc. 29th SIGIR*, pages 372–379, 2006.
[4] B. Bhattacharjee, S. Chawathe, V. Gopalakrishnan, P. Keleher, and B. Silaghi. Efficient peer-to-peer searches using result-caching. In *Proc. 2nd IWP2PS*, 2003.
[5] C. Buckley and A. F. Lewit. Optimization of inverted vector searches. In *Proc. 8th SIGIR*, pages 970–110, 1985.
[6] G. Das, D. Gunopulos, N. Koudas, and D. Tsirogiannis. Answering top-k queries using views. In *Proc. 32nd VLDB*, pages 451–462, 2006.
[7] R. Fagin. Combining fuzzy information from multiple systems. *JCSS*, 58(1):83–99, 1999.
[8] R. Fagin. Combining fuzzy information: An overview. *SIGMOD Record*, 31(2):109–118, 2002.
[9] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *JCSS*, 66(4):614–656, 2003.
[10] U. Güntzer, W.-T. Balke, and W. Kießling. Optimizing multi-feature queries for image databases. In *Proc. 26th VLDB*, pages 419–428, 2000.
[11] D. Harman and G. Candela. Retrieving records from a gigabyte of text on a mini-computer using statistical ranking. *JASIS*, 41(8):581–589, 1990.
[12] R. Kaushik, R. Krishnamurthy, J. Naughton, and R. Ramakrishnan. On the integration of structure indexes and inverted lists. In *Proc. SIGMOD*, pages 779–790, 2004.
[13] X. Long and T. Suel. Optimized query execution in large search engines with global page ordering. In *Proc. 29th VLDB*, pages 129–140, 2003.
[14] X. Long and T. Suel. Three-level caching for efficient query processing in large web search engines. In *Proc. 14th WWW*, pages 257–266, 2005.
[15] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.
[16] S. Nepal and M. V. Ramakrishna. Query processing issues in image (multimedia) databases. In *Proc. 15th ICDE*, pages 22–29, 1999.
[17] A. Ntoulas and J. Cho. Pruning policies for two-tiered inverted index with correctness guarantee. In *Proc. 30th SIGIR*, pages 191–198, 2007.
[18] M. Persin, J. Zobel, and R. Sacks-Davis. Filtered document retrieval with frequency-sorted indexes. *JASIS*, 47(10):749–764, 1996.
[19] R. Schenkel, A. Broschart, S. Hwang, M. Theobald, and G. Weikum. Efficient text proximity search. In *Proc. 14th SPIRE*, pages 287–299, 2007.
[20] A. Schrijver. *Combinatorial Optimization*. Springer, 2003.
[21] A. Soffer, D. Carmel, D. Cohen, R. Fagin, E. Farchi, M. Herscovici, and Y. S. Maarek. Static index pruning for information retrieval systems. In *Proc. 24th SIGIR*, pages 43–50, 2001.
[22] H. R. Turtle and J. Flood. Query evaluation: Strategies and optimizations. *IPM*, 31(6):831–850, 1995.
[23] H. E. Williams, J. Zobel, and D. Bahle. Fast phrase querying with combined indexes. *ACM TOIS*, 22(4):573–594, 2004.
[24] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes*. Morgan Kaufmann, 1999.
[25] W. Wong and D. Lee. Implementation of partial document ranking using inverted files. *IPM*, 29(5):647–669, 1993.
[26] J. Zhang and T. Suel. Efficient query evaluation on large textual collections in a peer-to-peer environment. In *Proc. 5th P2P*, pages 225–233, 2005.
[27] J. Zhang and T. Suel. Optimized inverted list assignment in distributed search engine architectures. In *Proc. 21st IPDPS*, pages 1–10, 2007.