Adaptive Algorithm for Threshold Path Subset Queries

Jérémy Barbay[1] and Aleh Veraskouski[2]

[1] DCC (Departamento de Ciencias de la Computación),
Universidad de Chile, Santiago, Chile.
jbarbay@dcc.uchile.cl
[2] CSCS (Cheriton School of Computer Science)
University of Waterloo, Canada.
averasko@cs.uwaterloo.ca
(graduated, now working for Amazon)

Abstract. In the context of queries to indexed search engines such as Google, Barbay and Kenyon [6] introduced and solved threshold set queries, answered by the set of references associated with at least t keywords out of the k given as input, for some constant parameter t. We sligthly generalize those results to the easy case where weights are associated to the keywords of the query, and to the more difficult case where weights are associated to the pairs of the relation between keywords and references. In the context of search queries on indexed file systems, Barbay *et al.* [5] introduced and solved path-subset queries, answered by the minimum set of subtrees which rooted path match all k keywords given as input. We combine both approaches to define and solve weighted threshold path-subset queries, answered by the minimum set of subtrees which rooted path match at least t keywords out of the k given as input, through the definition of a reduction to threshold queries.

Keywords. Adaptive algorithms, weighted threshold path-subset queries, multi-labeled tree.

1 Introduction

Consider the task of a search engine answering conjunctive queries: given a set of keywords, it must return a list of references to the objects relevant to all those keywords. These objects can be web-pages as in the case of a search engine such as `Google` or `Yahoo!`, documents as in a file system, or any other kind of data searched by keywords. A search engine typically uses a *precomputed index*, representing a binary relation between the set of n objects and the set of σ admissible keywords, or a labeled tree indexing a file system. Conjunctive queries are *schema-free* [12, 16]: they can be written without making any assumption about the structure of the document (e.g. its *schema* in XML [18] documents). This is important in applications where many documents with many different schemes must be searched [1].

Adaptive algorithms take advantage of "easy" instances, i.e. their run-time depends on some measure of the difficulty of the instance. Demaine *et al.* [8] considered some applications to queries on postings lists and studied adaptive algorithms for the union, intersection and difference of sets represented by sorted arrays.

We consider weighted queries on both weighted binary relations and weighted labeled trees, based on the definition of a score function on the objects of a binary relation or on the nodes of a tree. We propose adaptive algorithms for two types of weighted queries: on weighted binary relations and on weighted labeled trees with path non-increasing weights. For each result, we measure the complexity by the number of search and priority queue operations performed.

The rest of the article is organized as follows. We describe the results that we either use, generalize, or improve upon in Section 2, in two categories: data structures (Section 2.1) and algorithms (Section 2.2). We describe our results on weighted binary relations in Section 3, and our results on trees with weighted labels in Section 4. Section 5 gives a discussion of the results.

2 Previous Work and Extensions

Various algorithms have been proposed to solve unweighted schema-free queries on binary relations, labeled trees, and other data structures. We review some examples of the data structures considered, and which queries and algorithms our solutions extend.

2.1 Data Structures

A binary relation between two ordered sets, such as one associating labels with objects, can be encoded as a set of sorted arrays called *postings lists*. In this case, the answer to a conjunctive query is the intersection of the subsets corresponding to those arrays. A binary search finds the *insertion rank* [3] of a particular element in a sorted array of n_α elements in time $\mathcal{O}(\lg n_\alpha)$, and a straightforward variant can be used to search the positions of a set of δ increasing values in time $\mathcal{O}(\delta \lg(n_\alpha/\delta))$ [6].

A binary relation can also be encoded as a set of compressed bit-vectors [13, 15], supporting the search of the insertion rank of a particular element in constant time, at the price of space; or using less space at the cost of time [5]. Similarly, a priority queue can be implemented using various data structures, for instances based on sorted arrays or succinct encodings in the word-RAM model. While a trivial pointer-based tree structure with k elements will result in $\mathcal{O}(\lg k)$ comparisons per insertion or deletion in the worst-case, the more advanced structure described by Andersson and Thorup [2] has only $\mathcal{O}((\lg \lg k)^2)$ per insertion or deletion amortized.

Considering the variety of data structures that can be used to implement binary relations and priority queues, each of them with a different trade-off between the space used and the time required to search in it, we express the complexity of our algorithms in the number of search and priority queue operations performed, so that the complexity of the algorithm can be inferred for each data structure.

The same holds for labeled trees, such as XML documents or an index of a file system, as their encoding can be reduced to the encoding of the tree structure and of the binary relation associating the nodes in preorder to one or more label. Many efficient encodings are known for ordinal trees [10, 11, 14], and any encoding can be used to implement the binary relation and support the search for the first ancestor or the next descendant of a node x matching some label α [5].

2.2 Queries and Algorithms

Conjunctive queries are well known. Indeed, most search engines implement them. Given a list of labels (e.g. keywords), the answer consists of all the objects (e.g. webpage references) which are associated with all of the labels. Given an index such as described in the section above, solving a conjunctive query composed of k labels implies computing the intersection of k rows in a binary relation, which is a well studied problem [3, 4, 8, 9]

As an empty intersection can be an uninformative answer to a conjunctive query, we should consider other approaches. Researchers in information retrieval suggest a number of ways to deal

[3] The *insertion rank* of an element x in a set X is the rank (the linear order) of x in the set $X \cup \{x\}$.

with this problem. For example, one can relax both queries and document index in a number of different ways that are summarized by Bordogna and Pasi [7]. Barbay and Kenyon [6] proposed the adaptive algorithm to answer the query where for a given parameter t the answer consists of the references matching at least t of the k labels composing the query. Given an index such as described in the section above, solving this new type of query implies computing the *threshold set* of k rows in a binary relation, the set of objects associated with at least t labels among the k specified.

We extend further this type of query to *weighted threshold* queries, by considering weighted queries $Q : [\sigma] \rightarrow \{0, \ldots, \mu_Q\}$, where σ is the number of admissible keywords, and weighted binary relations $R : [\sigma] \times [n] \rightarrow \{0, \ldots, \mu_R\}$ [4]. The *score* of an object x relative to a query Q on a relation R is then defined as the linear combination of those weights, i.e. $\texttt{score}(R, Q, x) = \sum_{\alpha \in [\sigma]} Q(\alpha) R(\alpha, x)$, that corresponds to the notion of the *Retrieval Status Value* (or RSV) described by Bordogna and Pasi [7]. The answer to a query with parameter t is the set of objects with score at least t: this definition matches the original one from Barbay and Kenyon when each weight is either null or unitary (the *unweighted* case).

On labeled trees, one possible adaptation of the idea of conjunctive query is the *path-subset* query [5]. Given a set of k labels, the answer to such a query consists of the set of nodes whose path to the root matches all the labels and that do not have any ancestors with such a property. We extend this type of queries further to *weighted threshold path-subset* queries, by considering weighted queries and weighted binary relations between labels and nodes in a tree (see Section 4 for the formal definition).

3 Queries on Binary Relations

We propose an adaptive algorithm to answer weighted threshold queries on weighted binary relations. It generalizes the original algorithm proposed for threshold queries on binary relations in the unweighted case [6], and its analysis is based on similar concepts, formalized and extended to the weighted case.

Any algorithm answering weighted threshold queries has to check the correctness of its result, by certifying that each object in the answer set has score at least the threshold. Rather than considering each object separately (which would require time linear in the total number of possible objects), an algorithm must consider whole blocks of consecutive objects at once in order to achieve a sublinear complexity. We formalize this by the notion of *partition-certificate* of an instance: a partition $(I_i)_{i \in [\delta]}$ of the set $[n]$ of all objects, such that for any $i \in [\delta]$ either there is a set S of labels such that no object of I_i is associated with a label in S, and the maximum potential score of any of these objects $\mu_R \sum_{\alpha \notin S} Q(\alpha)$ is less than the t; or I_i is a single object $\{x\}$ (whose score can be larger or smaller than the threshold). The presence of singletons whose elements are not in the result set, is unavoidable in the weighted case, where an object can, for instance, be associated with all possible labels and still not score enough to be in the result set.

There are several ways to define the difficulty of an intersection instance, such as the minimal encoding size of a certificate [8], or the minimal number of comparisons [6]. We define the *alternation* of a weighted threshold set instance as the size δ of the smallest possible partition-certificate of the instance. The alternation is related to the *non-deterministic* complexity of the instance, as it corresponds to the complexity of a non-deterministic algorithm which would produce the shortest

[4] By $[m]$ we denote $\{1, \ldots, m\}$ for any integer number m.

partition-certificate of the instance. In the unweighted case (where all weights are unitary), if no object match the query then the alternation is exactly the non-deterministic complexity of the instance, i.e. the complexity of the best non-deterministic algorithm checking the answer to the query.

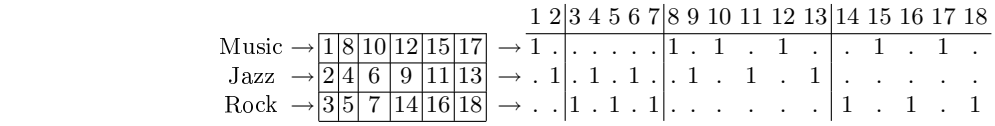

Fig. 1. An example of how a conjunctive query composed of three keywords corresponds to the intersection of the three corresponding sets. The alternation of the instance is $\delta = 4$, the number of intervals of a partition certificate where each interval has an empty intersection with at least one of the sets. Barbay and Kenyon's algorithm performs $7 \leq \delta k = 12$ searches (for the numbers $1, 2, 3, 8, 9, 14, 15$).

Barbay and Kenyon [6] proved that any randomized algorithm performs $\Omega(\delta k)$ searches in the worst case over instances of difficulty δ on k labels, and proposed an optimal deterministic algorithm for the unweighted case on sorted arrays. We analyze the complexity of the algorithms in terms of search and priority queue operations, where a priority queue operation is either an insertion or a deletion from a priority queue, and where each search operation is a search for the particular object in a data structure representing an ordered list of objects. We propose an optimal algorithm for the weighted case with any data structure supporting the search for the insertion rank in an indexed set:

Theorem 1. *Consider a weighted binary relation $R : [\sigma] \times [n] \to \{0, \ldots, \mu_R\}$, a weighted query $Q : [\sigma] \to \{0, \ldots, \mu_Q\}$, and a non-negative integer t. There is an algorithm that computes the threshold set for Q on R with threshold-value of t in $\mathcal{O}(\delta k)$ search and priority queue operations, where δ is the alternation of the instance and k is the number of labels of positive weight in Q.*

Algorithm 1 Algorithm answering Threshold Set queries

```
Set x to −∞, NO and YES to ∅ and MAYBE to the set of all labels of non-null weight;
Update(x, YES, NO, MAYBE, score_min, score_max) using Algorithm 2;
while x < ∞ do
    Set α to the next label from MAYBE in round robin order, and deduct μ_R Q(α) from score_max;
    Search for the insertion rank of x among the objects labeled α;
    if x is associated with a label α then
        Move α from MAYBE to YES;
        Add Q(α)R(α, x) to score_min and score_max;
        if t ≤ score_min then Output x;
    else
        Move α from MAYBE to NO;
    end if
    if t ≤ score_min or t > score_max then
        Update(x, YES, NO, MAYBE, score_min, score_max);
    end if
end while
```

Proof (of Theorem 1). Consider the steps of Algorithm 1: given a query Q with k positive weights and a threshold-value t, the algorithm computes the set of objects scoring at least t for a weighted binary relation R associating objects with labels.

Our algorithm goes through a number of phases. At each phase it considers one object x, in increasing order, and bounds its score by an interval $[\texttt{score\_min}, \texttt{score\_max}]$. The algorithm can decide whether x belongs to the result set through this interval and without computing the object's exact score ($t \leq \texttt{score\_min} \leq \texttt{score}(x)$). On the other hand, if for a given interval of consecutive objects there is a set of labels not associated with any of them with large total weight, this interval certifies that none of those objects belongs to the result set ($\texttt{score}(x) \leq \texttt{score\_max} < t$). The key issue of the algorithm is the choice of the values of x and of the labels to consider.

This choice is described in Algorithm 2, which is based on the decomposition of the set of labels of positive weights in three disjoint sets: `YES`, `MAYBE` and `NO`:

- `YES` corresponds to the labels already known to be associated with the current value of x. It can be implemented as a simple set, for instance in an array.
- `MAYBE` corresponds to the labels which could be associated with the current value of x. It is implemented as a FIFO queue so that each label in it is considered equally often.
- `NO` corresponds to the labels which are known not to be associated with the current value of x. It is implemented as a priority queue of at most k elements, and the labels α it contains are ordered by the value of the first object larger than x associated with label α.

The values of the bounds `score_min` and `score_max` on the potential score of x are direct consequences of those definitions: `score_min` depends on the weights of the labels in `YES`, i.e. $\texttt{score\_min} = \sum_{\alpha \in \texttt{YES}} Q(\alpha) R(\alpha, x)$; and `score_max` adds the maximum potential weights of the labels in `MAYBE` to `score_min`, i.e. $\texttt{score\_max} = \texttt{score\_min} + \sum_{\alpha \in \texttt{MAYBE}} Q(\alpha)\mu_R$.

To choose a new value for x, the algorithm removes labels from the set `MAYBE` till it reaches a critical weight, where removing any other label would make it impossible for an object matching only the labels of `MAYBE` to score above the threshold. Then, the smallest object potentially in the result set corresponds to the first label of the priority queue implementing set `NO`.

Consider a phase of the execution where the algorithm is processing an interval of the partition-certificate consisting of only one object x. Algorithm 1 performs at most k iterations of the main loop to decide whether x has enough score or not without updating x (through Algorithm 2). Once the decision about x is made, the algorithm updates x and moves to the next phase. Updating of x takes not more than k loop iterations of Algorithm 2. Thus during each phase, the algorithm performs at most $\mathcal{O}(k)$ search and priority queue operations.

Consider a phase corresponding to the interval of the partition-certificate that does not have any objects with enough score and a subset S of labels that are not associated with any of the objects in this interval. Algorithm 1 may update x more than once during the same phase. We prove the upper bound on the number of operations through considering the way the algorithm moves labels from one set to another.

The only way Algorithm 1 moves labels is from set `MAYBE` to either set `YES` or set `NO`. Algorithm 2, on the other hand, move labels from `YES` to `MAYBE`, from `MAYBE` to `NO`, and from `NO` to `YES` in this order. As it cannot move labels that are in S to `YES`, the algorithm has the only possible loop `MAYBE` $\longrightarrow$ `NO` $\longrightarrow$ `YES` $\longrightarrow$ `MAYBE` for these labels.

However, the algorithm does not move any labels from S that it already moved to `NO` during the processing of the same interval, because the label's successor is out of the current interval and cannot be processed in the current phase. While the algorithm retrieves labels from set `MAYBE` in

round-robin order, it cannot retrieve any label from set `MAYBE` for the second time until all the labels from subset S appear in set `NO`, which effectively means that the next element x will be outside of the interval and the algorithm proceeds to the new phase. While it takes a constant time for the algorithm to move each label from set `MAYBE` back to set `MAYBE`, it needs $\mathcal{O}(k)$ search and priority queue operations to complete this phase.

As the algorithm spends $\mathcal{O}(k)$ to complete any phase, and any instance has δ intervals that correspond to δ phases, the total complexity of the algorithm is $\mathcal{O}(\delta k)$. □

Algorithm 2 `Update`$(x, \texttt{YES}, \texttt{NO}, \texttt{MAYBE}, \texttt{score\_min}, \texttt{score\_max})$

Add all the labels from `YES` to the set `MAYBE` and set `score_max` to $\sum_{\alpha \in \texttt{MAYBE}} Q(\alpha)\mu_R$;
Choose a label α in round-robin order from `MAYBE`;
while `score_max` $- Q(\alpha)\mu_R \geq t$ **do**
  Deduct $Q(\alpha)\mu_R$ from `score_max`, and move α from `MAYBE` to `NO`;
  Choose a label α in round-robin order from `MAYBE`;
end while
Find the subset $S \subset$ `NO` of labels α such that the successor of x among the objects labeled α is minimal;
Move all the labels of S from `NO` to `YES`, and set `score_min` to $\sum_{\alpha \in \texttt{YES}} Q(\alpha)R(\alpha, x)$;
Update x to its successor among the objects labeled α, for any label in `YES`;

Note that k is the number of labels with a positive weight (i.e. non-null). If the binary relation is implemented by postings lists, and the priority queue is implemented using a heap, the complexity of the algorithm is $\mathcal{O}(\delta k \lg(n/(\delta k)) + \delta k \lg k)$, where n is the sum of the sizes of all postings lists and k is the maximum size of the priority queue. If the binary relation is implemented using Barbay *et al.*'s [5] succinct encoding and the priority queue is implemented using Andersson and Thorup's [2] structure, the complexity of the algorithm is $\mathcal{O}(\delta k \lg\lg\sigma + \delta k (\lg\lg k)^2)$ in the RAM model with word size $\Theta(\lg\max\{\sigma, n\})$.

4 Queries on Labeled Trees

The main idea of path-subset queries [5] is that the effect of labels associated with nodes "propagates" to the descendants of nodes. We extend this concept through the definition of a score function on the nodes of the tree that depends on the labels associated with a node and its ancestors, and on the weight of these associations.

Formally, given a query Q on a tree T labeled through the relation R, the path-score of a node x is defined as the sum of maximum values of $Q(\alpha)R(\alpha, y)$ for each node y which is x or one of its ancestors, over all labels $\alpha \in [\sigma]$. Each label is counted only once, i.e. a label α contributes only $\max_y R(\alpha, y)$ to node x, where y is x or one of its ancestor. This defines the *path-score* of x as

$$\texttt{path\_score}(T, R, Q, x) = \sum_{\alpha \in [\sigma]} Q(\alpha) \max_{y \in \texttt{ancestors}(x) \cup \{x\}} R(\alpha, y).$$

Combining this score function on nodes with the concept of weighted threshold set queries in the context of weighted labeled trees brings the concept of *weighted threshold path-subset* queries,

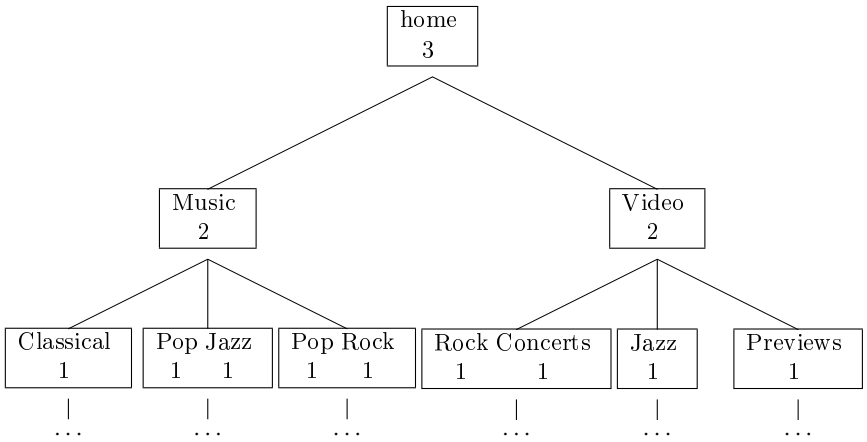

Fig. 2. An example of a simple file system. Each node represents a folder and contains the words associated with it, along with the weight of these associations.

answered for a given parameter t by the set of nodes of path-score at least t that do not have any ancestor matching this property.

We propose an algorithm to solve these queries in the case where the labels are associated with the nodes on the same root-to-leaf path with non-increasing weights, i.e. there is no such a node x that has a label α associated with it with some weight $R(x, \alpha)$ and that has a descendant x' associated with the same label with larger weight $R(x', \alpha) > R(x, \alpha)$. This non-increasing restriction does not restrict instances where the weights of the labels of the tree are all null or unitary: in both cases trees are non-increasing by definition.

This restriction makes the contribution of a label α to the path-score of a node x depend only on the weight of the closest to the root ancestor of the node x associated with the label α, instead of depending on the arbitrary one with the large weight of its association with the label α. To solve weighted threshold path-subset queries in the general case, an algorithm would have to compute $\max_{y \in \texttt{ancestors}(x) \cup \{x\}} R(\alpha, y)$ regularly, which makes it more complex.

We describe an adaptive analysis of the complexity of our algorithm by using a measure of difficulty inspired by the partition-certificates and alternation, as defined for queries on binary relations. As before, any algorithm answering a weighted threshold path-subset query has to check the correctness of its result. For this query-type, it corresponds to producing a certificate that each node in the answer set has a path-score of at least the threshold, and that each node that is not in the answer set either has an ancestor that is in this set or has a path-score smaller than the threshold.

Any order of the nodes can be used to easily define sets of nodes that cannot belong to the answer set. As threshold path-subset queries are based solely on the ancestor-descendant relation between nodes, we propose an analysis based on the preorder traversal of the tree, in which all the descendants of a node are consecutive. As Figure 2 represents an example of the file system with nodes corresponding to files and folders and labels corresponding to their names, Figure 3 represents the binary encoding of it.

We generalize the concept of the *partition-certificate*, introduced on binary relations, to multi-labeled trees as a partition $(I_i)_{i \in [\delta]}$ of the set $[n]$ of all nodes, such that for any $i \in [\delta]$ either

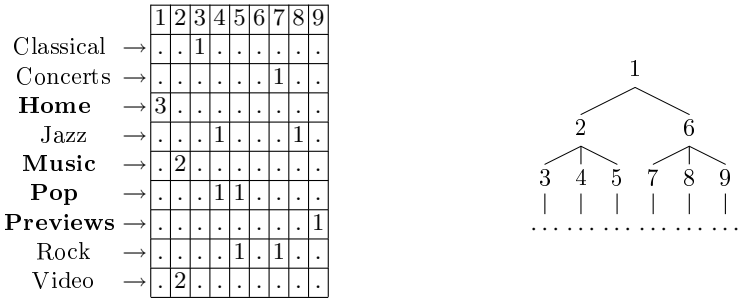

Fig. 3. The encoding of the example of Figure 2 using a weighted binary relation. The null weights are noted by dots for the sake of readability. Each number in the schema of the tree is the preorder rank of the corresponding node.

(i) I_i corresponds exactly to a subtree with a root x, such that the path-score of x is at least the threshold and each ancestor of x has a path-score lower than the threshold; or

(ii) there is a set S of labels such that no label from S is associated with any node in I_i or any of its ancestors, and such that the sum of the maximum possible weights of the remaining labels is insufficient to reach the threshold-value: $\sum_{\alpha \notin S} Q(\alpha)\mu_{R} < t$; or

(iii) all the elements in I_i have path-subset smaller than threshold but are not in (ii), i.e. they do not have a subset S of labels with the properties described.

In the first case, I_i corresponds to a subtree such that the path-score of the root x is at least the threshold, so that x is in the result set and all its descendant can be ignored. In the second case, I_i corresponds to a block of consecutive nodes in preorder that do not match enough labels to have sufficient weight, even assuming that all other labels contribute maximum possible value μ_{R} to their path-score. In the third case, I_i consists of node(s) whose path-score is less than the threshold as in the second case, but that do not have a subset of labels mentioned above, i.e. they would have gotten path-score of threshold or more, if all the labels associated with them or their root path had contributed μ_{R} each.

As for binary relations, we define the *alternation* as the size δ of the smallest possible partition-certificate of the instance and use it to analyze the complexity of our algorithm.

If we consider the weighted tree at Figure 3, the weighted query of Figure 4 with a threshold-value $t = 5$, and $\mu_{R} = 3$, we get the minimal partition-certificate shown at Figure 5. This partition-certificate contains all three possible types of intervals. The interval $\{2, \ldots, 5\}$ is the whole subtree with the root $\{2\}$ that has enough path-score: $\texttt{path\_score}(2) = 3 \times 1 + 2 \times 2 = 7 > 5$. The intervals $\{1\}$ and $\{6, \ldots, 8\}$ are intervals that have a subset of labels $S = \{\text{Music}, \text{Pop}, \text{Previews}\}$ not associated with any node and that is large enough to guarantee that no nodes can have path-score of at least t: $\mu_{R} \sum_{\alpha \notin S} Q(\alpha) = 3 \times 1 = 3 < 5 = t$. And the interval $\{9\}$ has a set $S \in \{\text{Music}, \text{Pop}\}$ that is not large enough, but whose single node does not have enough weight either.

Barbay *et al.* [5] proved that any randomized algorithm performs $\Omega(\delta k)$ search operations in the worst case over (unweighted) path subset queries of k labels and of alternation δ. This is a particular case of weighted threshold path-subset queries, where $\mu_{R} = \mu_{Q} = 1$ and where the threshold-value t is the number k of labels α of non-null weight $Q(\alpha)$. We propose an optimal algorithm for the cases with arbitrary values for μ_{R}, μ_{Q} and t, restricted only in the weights assigned to labels in the multi-labeled tree:

Theorem 2. *Consider a tree T, a weighted binary relation $R : [\sigma] \times [n] \to \{0, \ldots, \mu_{R}\}$ assigning path non-increasing weighted labels to the nodes of T, a weighted query $Q : [\sigma] \to \{0, \ldots, \mu_{Q}\}$, and*

Keywords: ($\alpha \in Q$)	Home	Music	Pop	Previews
Weights: ($Q(\alpha)$)	1	2	1	1

Fig. 4. An example of the weighted conjunctive query of 4 words.

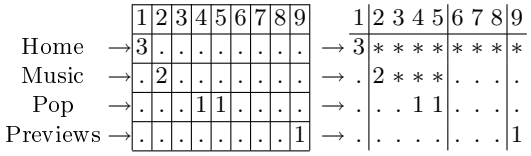

Fig. 5. An example of the minimal partition-certificate for the weighted tree and weighted query described above and the $t = 5$. This partition-certificate has all three possible types of intervals. The alternation of the instance is $\delta = 4$, the number of intervals of a partition-certificate.

a non-negative integer t. There is an algorithm that computes the threshold set for Q on T and R with threshold-value of t in $\mathcal{O}(\delta k)$ search and priority queue operations, where δ is the alternation of the instance and k is the number of labels of positive weight in Q.

Proof (of Theorem 2). Consider the steps of Algorithm 3: given a query Q with k positive weights and a threshold-value t, the algorithm computes the set of the highest nodes with a path-score of at least t in a tree T labeled by a weighted binary relation R.

The algorithm proceeds along the nodes of the tree in increasing order (according to the preorder defined on the tree). At each phase it considers a node x and computes the minimum possible path-score `score_min` and the maximum possible path-score `score_max` for it. If `score_min` $\geq t$, it is in the threshold subset path. The algorithm puts it to the output and starts the next phase by proceeding to the first successor of x that is not one of its descendants, i.e. the algorithm skips the whole subtree rooted with the node x. If $t >$ `score_max` the node x is guaranteed to have the path-score smaller than the threshold, thus the algorithm proceeds to the next node in the tree that might be in the threshold subset path finishing the current phase as well.

The decision about the current node is made based on the division of the set of labels of positive weights into three disjoint sets: `YES`, `MAYBE` and `NO`.

- `YES` consists of the labels already known to be associated with the current node x or one of its ancestors.
- `MAYBE` consists of the labels that we do not know yet whether they are associated with the current node x or one of its descendant or not. This set is implemented as a queue so that each label in it is retrieved equally often.
- `NO` consists of the labels that are known not to be associated with the current node x nor any of its ancestors. This set is implemented by a priority queue with at most k elements, and the labels in it are ordered by the preorder number of the first α-successor x_α of the current node x.

The values of `score_min` and `score_max` here depend not only on the labels assigned to x, but also on the labels assigned to its ancestors, and are computed as `score_min` $= \sum_{\alpha \in \texttt{YES}} Q(\alpha) \max_{y \in \texttt{ancestors}(x) \cup \{x\}} R(\alpha, y)$ and `score_max` = `score_min` $+ \sum_{\alpha \in \texttt{MAYBE}} Q(\alpha)\mu_R$.

After making a decision about the node x, Algorithm 3 updates the node (through Algorithm 4) and finds the next node x to advance to. It performs the search for the next node x in the similar to the case of binary relations way, except that now it should move some labels from set `NO` to set `MAYBE` as well, because the node x might have been already increased by Algorithm 3, in the case of

the phase where the algorithm is processing an interval consisted of the whole subtree with a root node in the answer set.

The complexity analysis of the algorithm is very similar to the one provided in Theorem 1. We consider the algorithm at each phase and how it processes each type of intervals in the minimal partition-certificate, and prove that for each phase the algorithm does at most $\mathcal{O}(k)$ search and priority queue operations. While the number of intervals is δ, we come up with the total complexity of $\mathcal{O}(\delta k)$. □

Algorithm 3 Algorithm answering Threshold Path-Subset queries

Set x to $-\infty$, `NO` and `YES` to $\emptyset$ and `MAYBE` to the set of all labels of non-null weight;
`Update`$(x, \texttt{YES}, \texttt{NO}, \texttt{MAYBE}, \texttt{score\_min}, \texttt{score\_max})$ using Algorithm 4;
while $x < \infty$ **do**
  Set α to the next label from `MAYBE` in round robin order, and deduct $\mu_R Q(\alpha)$ from `score_max`;
  if x or one of its ancestors is labeled α **then**
    Move α from `MAYBE` to `YES`;
    Find y, the closest to the root ancestor of x with the label α;
    Add $Q(\alpha)R(\alpha, y)$ to `score_min` and `score_max`;
    if $t \leq$ `score_min` **then**
      Output x;
      Update x to its first preorder successor which is not one of its descendants;
    end if
  else
    Move α from `MAYBE` to `NO`;
  end if
  if $t \leq$ `score_min` or $t >$ `score_max` **then**
    `Update`$(x, \texttt{YES}, \texttt{NO}, \texttt{MAYBE}, \texttt{score\_min}, \texttt{score\_max})$;
  end if
end while

Algorithm 4 `Update`$(x, \texttt{YES}, \texttt{NO}, \texttt{MAYBE}, \texttt{score\_min}, \texttt{score\_max})$

Move all the labels from `YES` to the set `MAYBE`;
Move each label α from `NO` that has α-successor less than current node x to the set `MAYBE`;
Set `score_max` to $\sum_{\alpha \in \texttt{MAYBE}} Q(\alpha)\mu_R$;
Choose a label α in round-robin order from `MAYBE`;
while `score_max` $- Q(\alpha)\mu_R \geq t$ **do**
  Deduct $Q(\alpha)\mu_R$ from `score_max`, and move α from `MAYBE` to `NO`;
  Choose a label α in round-robin order from `MAYBE`;
end while
Find the subset $S \subset$ `NO` of labels α such that the preorder successor of x among the nodes labeled α is minimal;
Move all the labels of S from `NO` to `YES`, and set `score_min` to $\sum_{\alpha \in \texttt{YES}} Q(\alpha) \max_{y \in \texttt{ancestors}(x) \cup \{x\}} R(\alpha, y)$
Update x to its preorder successor among the nodes labeled α, for any label of `YES`;

5 Discussion

In the context of the search in binary relations, such as the one associating labels with objects (e.g. keywords with webpages), we identified the intuition behind previous work on threshold set queries

in binary relations [6] and applied it in much more general contexts: where weights are associated with the terms of the query and with the relation between objects and labels.

In the context of the search in a multi-labeled tree of unknown schema, such as one representing the index of a file system, we applied the threshold set concept to path-subset queries [5]. In both contexts we define queries which are more informative than the queries previously considered, while being not substantially more expensive to answer.

The concept of weighted threshold set queries can be applied to some other type of schema-free queries on multi-labeled trees, among which we describe three in particular:

- *additive path-subset queries*, which are similar to path-subset queries but with a different score function, where for each node x the contribution of its ancestors labeled α adds up to form its score;
- *path-subsequence queries*, which are similar to path-subset queries but with a required order on the labels of the query (obviously, the path-score can then be defined in an additive or non-additive way);
- *labeled lowest common ancestor*, where the descendants of a node contribute to its score, rather than its ancestors (the concept was already defined without threshold nor weights [12, 16, 17]).

As threshold set queries generalize conjunctive queries, for which many algorithms have been studied [3, 4, 8, 9], many other algorithms should be considered, some of which could take advantage of properties of the instances other than those described by the *alternation* measure of difficulty.

Although the adaptive analysis is finer than a typical worst case analysis, its value for a particular application depends of the appropriateness of the corresponding difficulty measure: some experimentations will be necessary. It is reasonably easy to generate a weighted index, for instance by assigning different weights to the labels associated with the links to a webpage, in the title or in a simple paragraph. It will be harder to generate realistic user queries for the threshold set: the users usually use conjunctive queries and adapt their queries to the type of results returned. In particular, they give a small number of keywords to avoid receiving a null answer. One solution is to consider conjunctive queries *extended* with some labels of small weight, corresponding to the profile of the user: such queries would help to adjust the answer to the initial conjunctive query based according to the user preferences.

References

[1] S. Amer-Yahia, S. Cho, and D. Srivastava. Tree pattern relaxation. In *Extending Database Technology*, pages 496–513, 2002.

[2] A. A. Andersson and M. Thorup. Tight(er) worst-case bounds on dynamic searching and priority queues. In *STOC '00: Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 335–342, New York, NY, USA, 2000. ACM Press.

[3] R. A. Baeza-Yates. A fast set intersection algorithm for sorted sequences. In *Proceedings of the 15th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 3109 of *Lecture Notes in Computer Science (LNCS)*, pages 400–408. Springer, 2004.

[4] R. A. Baeza-Yates and A. Salinger. Experimental analysis of a fast intersection algorithm for sorted sequences. In *Proceedings of 12th International Conference on String Processing and Information Retrieval (SPIRE)*, pages 13–24, 2005.

[5] J. Barbay, A. Golynski, J. I. Munro, and S. S. Rao. Adaptive searching in succinctly encoded binary relations and tree-structured documents. *ELSEVIER Theoretical Computer Science (TCS)*, October 2007.

[6] J. Barbay and C. Kenyon. Adaptive intersection and t-threshold problems. In *Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 390–399. Society for Industrial and Applied Mathematics (SIAM), January 2002.

[7] G. Bordogna and G. Pasi. Modeling vagueness in information retrieval. In M. Agosti, F. Crestani, and G. Pasi, editors, *ESSIR*, volume 1980 of *Lecture Notes in Computer Science*, pages 207–241. Springer, 2000.

[8] E. D. Demaine, A. López-Ortiz, and J. I. Munro. Adaptive set intersections, unions, and differences. In *Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 743–752, 2000.

[9] E. D. Demaine, A. López-Ortiz, and J. I. Munro. Experiments on adaptive set intersections for text retrieval systems. In *Proceedings of the 3rd Workshop on Algorithm Engineering and Experiments*, volume 2153 of *Lecture Notes in Computer Science (LNCS)*, pages 5–6, Washington DC, January 2001.

[10] R. F. Geary, R. Raman, and V. Raman. Succinct ordinal trees with level-ancestor queries. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1–10, 2004.

[11] G. Jacobson. Space-efficient static trees and graphs. In *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 549–554, 1989.

[12] Y. Li, C. Yu, and H. V. Jagadish. Schema-free XQuery. In *VLDB*, 2004.

[13] D. Okanohara and K. Sadakane. Practical entropy-compressed rank/select dictionary. In *Proceedings of the 14th International Workshop on Algorithms and Data Structures (WADS)*, volume 1671 of *Lecture Notes in Computer Science (LNCS)*. Springer-Verlag, 2007.

[14] R. Raman, V. Raman, and S. S. Rao. Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete algorithms*, pages 233–242, 2002.

[15] K. Sadakane and R. Grossi. Squeezing succinct data structures into entropy bounds. In *Proceedings of the 17th annual ACM-SIAM symposium on Discrete algorithm*, pages 1230–1239, 2006.

[16] A. Schmidt, M. L. Kersten, and M. Windhouwer. Querying XML documents made easy: Nearest concept queries. In *ICDE*, pages 321–329, 2001.

[17] Y. Xu and Y. Papakonstantinou. Efficient keyword search for smallest LCAs in XML databases. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 527–538, New York, NY, USA, 2005. ACM Press.

[18] F. Yergeau, T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. Extensible Markup Language (XML) 1.0 (third edition). Technical report, W3C Recommendation, February 2004.