Distributed algorithm for Ranking under Temporal Constraints

Aman Sawhney, Lena Mashayekhy,
Department of Computer Science
University of Delaware
Newark, DE 19716, USA
{asawhney, mlena}@udel.edu

***Abstract*—Search engines order documents by relevance to retrieve results in response to user queries. Relevance of results is evaluated using ranking functions, which are composed of feature and associated feature weights. In response to a query, features values for each query-document pair are generated and input to a ranking function, which then computes a score for each pair. This score is used to generate an ordered subset of documents.**
The notion of ranking documents under temporal constraints, ie. to produce a ranked order of documents given a time constraint(T(q)), has been around for a while. In order to satisfy T(q), only a subset of all features may be considered while evaluation.Naturally, this ranked order might not be best possible. An intuitive modification to improve performance is to model ranking functions in a distributed setting and, leverage parallelism to use more features within the same time constraint.
This paper models temporally constrained ranked functions in a distributed setting and provides a greedy algorithm to rank a document corpus(D) in response to a query(q) within a time constraint(T(q)). We use simulations to study the performance of our algorithm in the special case of homogeneous machines.

***Keywords*-Distributed, Algorithm, Information Retrieval**

I. Introduction

Our Contribution.

Related Work. [1]

Organization.

II. Wang et. al Indep model

Let q be the query placed by a user, d be the set of documents in the index, F be the feature set, where

$$F = \{f_1,\ f_2,\ \ldots,\ f_n\}$$

then

$$score(q,\ d)\ = \sum_i (\lambda_i . f_i(q,\ d))$$

where λ_i is the weight associated with feature f_i
The features f_i can be either term features(unigram) or term proximity features(bigrams).

If the model is temporally constrained with a limit of $T(q)$, i.e. the time associated with the query q then

$$score(q,\ d)\ = \sum_{f_i(q,\ .):\ S_i\ =1} (\lambda_i . f_i(q,\ d))$$

s.t.

$$\sum_{f_i(q,\ .):\ S_i\ =1} C(f_i(q,\ .))\ <= T(q)$$

Here, $C(f_i(q))$ is the cost associated with feature i. $S_i\ = 1$ implies that the feature i is selected.

$$\hat{S} = \ argmax_S\ \sum_i S_i\ .\ \lambda_i(q)$$

s.t.

$$\sum_i S_i\ C(f_i(q,\ .))\ <= T(q), \quad S_i\ \in \{0,\ 1\}$$

Cost is computed by the document frequency that the query generates for the a particular feature.

So, in this model there is a profit(λ) associated with the selection of a feature, a weight($C(f)$) that has to be borne and a total weight constraint(time constraint $T(q)$). Hence, the problem can viewed as a classic *knapsack problem*. The authors solve the problem using a greedy approach: sort the features in decreasing ratio of profit to weight density, add features one by one until you exceed the time constraint or run out of of features.

III. Problem

In this section, we model the problem of temporally constrained ranking with multiple processors, where there is a time constraint $T(q)$ for a query q.

Linear ranking functions are a class of effective ranking models used for information retrieval. A linear ranking function is a characterized by a set of features $F = \{f_1, \ldots, f_N\}$ and their corresponding model parameters $\Lambda = \{\lambda_1, \ldots, \lambda_N\}$, where each feature f_i is a function mapping a query-document pair (q, d) to a real value. The relevance of document d to query q, $score(q, d)$, is defined as a weighted linear combination of the feature values computed over the document and the query. As a result,

$score(q, d)$ is defined as follows:

$$score(q, d) = \sum_{i=1}^{N} \lambda_i(q) f_i(q, d) \quad (1)$$

where $\lambda_i(q)$ is calculated based on meta-features defined over query q for each feature i, and other

Since using all features in the linear ranking function may exceed the time constraint, we need to change the efficiency characterises of the ranking function by choosing only a subset of the features in order to meet its time requirement. The resulting linear ranking function is a temporally constrained linear ranking function.

The problem can be formulated as an integer program, as follows:

$$\text{Maximize } score(q, d) = \sum_{i=1}^{N} \sum_{p=1}^{P} \lambda_i(q) \cdot x_{ip} \quad (2)$$

Subject to:

$$\sum_{p=1}^{P} x_{ip} \leq 1, \forall i = 1, \ldots, N \quad (3)$$

$$\sum_{i=1}^{N} w_i C(f_i(q, \cdot)) x_{ip} \leq T(q), \forall p = 1, \ldots, P \quad (4)$$

$$x_{ip} \in \{0, 1\}, \forall i = 1, \ldots, N, \forall p = 1, \ldots, P \quad (5)$$

The decision variables x_{ip} are defined as follows:

$$x_{ip} = \begin{cases} 1 & \text{if } f_i \text{ is allocated to } p, \\ 0 & \text{otherwise.} \end{cases} \quad (6)$$

where P is the number of computers and N is the number of features.

The objective function is to maximize the total score of the selected features. Constraints (3) ensure that each feature is selected at most once. Constraints (4) guarantee that the computational cost of selected features for each processor does not exceed the time requirement for any processor. Constraints (5) represent the integrality requirements for the decision variables.

Notice that the time constraint for all P machines is the same ($T(q)$) because these can all be run in parallel.
This can be modeled in a similar greedy manner.

A. Homogeneous Machines

Murthy et. al [2] provide an algorithm that solves the multiple knapsack problem in a distributed manner. An adaptation of that algorithm for our problem follows.

Algorithm 1 Greedy Distributed (homogeneous machines) algorithm with $T(q)$ time constraint for each machine

1: $ItemList \leftarrow ItemList.SortDecreasingBy(\frac{\lambda(f_i,.)}{C(f_i,.)})$
2: i=0
3: $\forall j, r_j = T(q)$
For the source p_s
4: **for** feature f_i; i=1 to n **do**
5: Broadcast $\langle C(f_i, .)\rangle$ to all p_j
6: Recieve$\langle j\rangle$
7: Assign i to j
8: **end for**
For the processor p_j**:**
9: Receive $\langle C(f_i, .)\rangle$ from p_s
10: **if** $r_j >= C(f_i)$ **then**
11: Broadcast $\langle j,\ r_j\rangle$ to all p'_j
12: **else**
13: Broadcast $\langle j,\ \perp\rangle$ to all p'_j
14: **end if**
15: Receive $\langle j, r_j\rangle\ from\ all\ p'_j$ best = $argmax_{j'}(r'_j)$ from p_s
16: **if** best = j **then**
17: Send $\langle j\rangle$ to all p
18: $r_j \leftarrow r_j - C(f_i, .)$
19: **end if**
procedure Final():
20: **for** j =1 to m **do**
21: Pick $max(p_j, max_{i:C(f_i)<=T(q)}(\lambda_i))$
22: **end for**

Homogeneous setting implies that the performance of the machines for the same input is similar(same ideally). As described in Algorithm 2, there is constraint on the computation time of all machines($T(q)$). In decreasing weight to cost ratio, the source broadcasts the cost(time) of computing the current feature to all machines. If the remaining computation time of a machine is greater than the cost of the feature, then the machine broadcasts its willingness to accept to all the machines. Since, all willing machines transmit their offer to all machines, whoever made the best offer claims the feature. This is continued till all features are finished or all machines are fully allocated. In this light, Algorithm 2 can viewed as a scheduling algorithm.

1) Toy Simulation: Let P be the number of machines and N be the number of features. It is reasonable to assume that :

$$N > P$$

The inputs to simulation are :

- Number of features N.
- A list of feature weights $\lambda(f_i)$ where $i \in [1..N]$.
- A list of feature costs $C(f_i)$ where $i \in [1..N]$.

- Number of machines P.
- Time constraint $T(q)$.

Scheduling

The machines are implemented as a matrix M with all values initialized as $T(q)$ ie.

$$M = [T(q),\ ,T(q),\ \ldots,\ T(q)]$$

of dimension $1 \times P$.

A decreasing list of weight to cost ratios (R) is calculated. Iterating over the list R (ie. all the features f_i s.t $i \in [1..N]$), the machine j with the maximum available computation time m_j, s.t $c_i \leq m_j$ is chosen to run feature f_i. In other words, for all the machines j ($j \in [1..P]$), whose current available computation time is greater than the cost of the feature i, choose the one with the maximum available time to run the feature f_i .

The value of j is then updated to be $m_j - c_i$. In case, there are multiple machines with same available computation time, a random choice is made. This is continued, until either all the features are assigned or all machines are fully allocated (including cases where some machines have time left but not enough to allocate remaining features).

The number of messages that are passed is computed implicitly(counting assuming that implementation is truly distributed). After the assignment is completed, the assignments that were made are reported along with the total number of messages passed and the utilization for each machine.

Since, the machines are homogeneous, given that all the input conditions are the same, the *makespan* would not change on any number of iterations of the algorithm.

A toy example of the simulation is presented in figure **??**.

B. heterogeneous machines

In the case of heterogeneous machines the computation time or the cost of a feature would be different for different machines. As such an assumption is made that an expected time completion matrix(ETC) that contains the expected completion times of any task on any machine is known to us. So,

$$ETC = \begin{Bmatrix} c_{11} & c_{12} & \ldots c_{1N} \\ \vdots & \vdots & \vdots \\ c_{P1} & c_{P2} & \ldots c_{PN} \end{Bmatrix}$$

where N is the number of features and P is the number of machines.

ETC along with λ can then be used to compute the weight to cost ratio matrix which essentially provides the ratio of weight and the cost incurred for a feature on a given machine.

$$W = \begin{Bmatrix} w_{11} & w_{12} & \ldots w_{1N} \\ \vdots & \vdots & \vdots \\ w_{P1} & w_{P2} & \ldots w_{PN} \end{Bmatrix}$$

where $w_{ij} = \frac{\lambda(f_i)}{c_{ij}}$

As before, $M = [T(q),\ ,T(q),\ \ldots,\ T(q)]$ with dimensions

Input configuration

$$N = 5$$
$$P = 2$$
$$\lambda(f_i) = \{10, 25, 55, 23, 34\}$$
$$C(f_i) = \{1, 2, 3, 4, 5\}$$
$$T(q) = 5$$

Then, Intialize $M = [5, 5]$ and feature assignment matrix $A = [[], []]$

$$R = \{10.0, 12.5, 18.33, 5.75, 6.8\}$$
$$argsorted\ R = \{2, 1, 0, 4, 3\}$$

f_2 has the highest weight/cost value and has cost =3.

Assign f_2 to a machine with $m_j > 2$
Since $m_1 = m_2 = 5$, pick at random
$A_2 \leftarrow f_2$

$$argsorted\ R = \{1, 0, 4, 3\}$$
$$A = [[], [2]]$$
$$M = [5, 2]$$

Next, $C(f_1) = 2$,
Assign f_1 to a machine with $m_1 > 2$
$A_1 \leftarrow f_1$

$$argsorted\ R = \{0, 4, 3\}$$
$$A = [[1], [2]]$$
$$M = [3, 2]$$

Next, $C(f_0) = 1$, Assign f_0 to a machine with $m_1 > 2$
$A_1 \leftarrow f_0$

$$A = [[1, 0], [2]]$$
$$M = [2, 2]$$
$$argsorted\ R = \{4, 3\}$$

Next,

$$C(f_4) = 5$$

Time required greater than available on any machine $break$.

Figure 1: Toy simulation

$1 \times P$.

Algorithm 2 Rough algorithm

1: Sort W in decreasing order
2: For all entries in W such that w_{ij} is not greater than T(q) and f_i is not assigned to any machine start machine assignment
3: Follow algorithm1 from line 4 onwards

The complexity of this algorithm would be $\mathcal{O}NP$

REFERENCES

[1] L. Wang, D. Metzler, and J. Lin, "Ranking under temporal constraints," in *Proceedings of the 19th ACM international conference on Information and knowledge management*. ACM, 2010, pp. 79–88.

[2] A. Murthy, C. Yeshwanth, and S. Rao, "Distributed approximation algorithms for the multiple knapsack problem," *arXiv preprint arXiv:1702.00787*, 2017.