Early Exit Optimizations for Additive Machine Learned Ranking Systems

B. Barla Cambazoglu, Hugo Zaragoza, Olivier Chapelle
Yahoo! Research

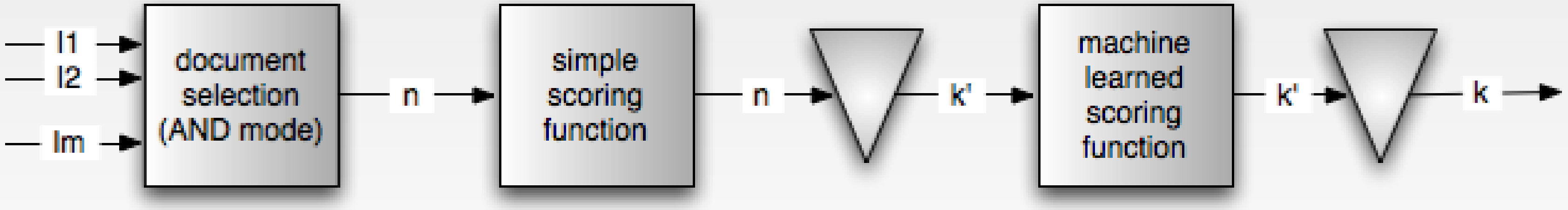
Jiang Chen, Ciya Liao, Zhaohui Zheng, Jon Degenhardt
Yahoo! Labs

Outline

- Early exit problem
- Heuristics
- Performance evaluation
- Open problems

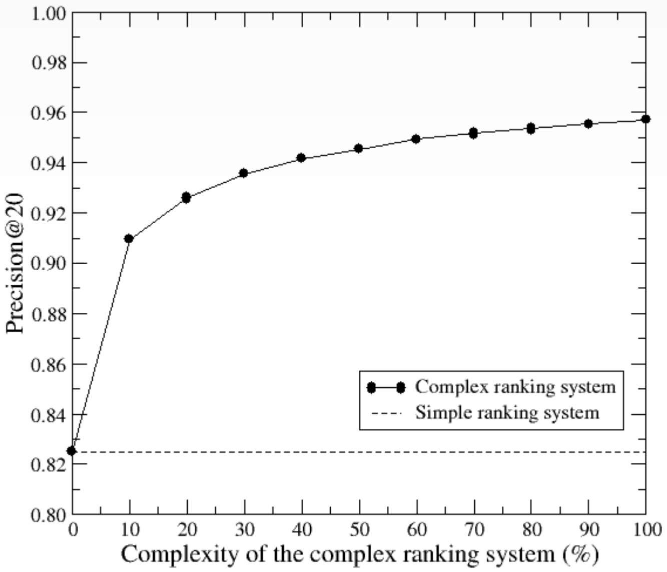
Ranking Architecture

- We consider a three-level ranking architecture



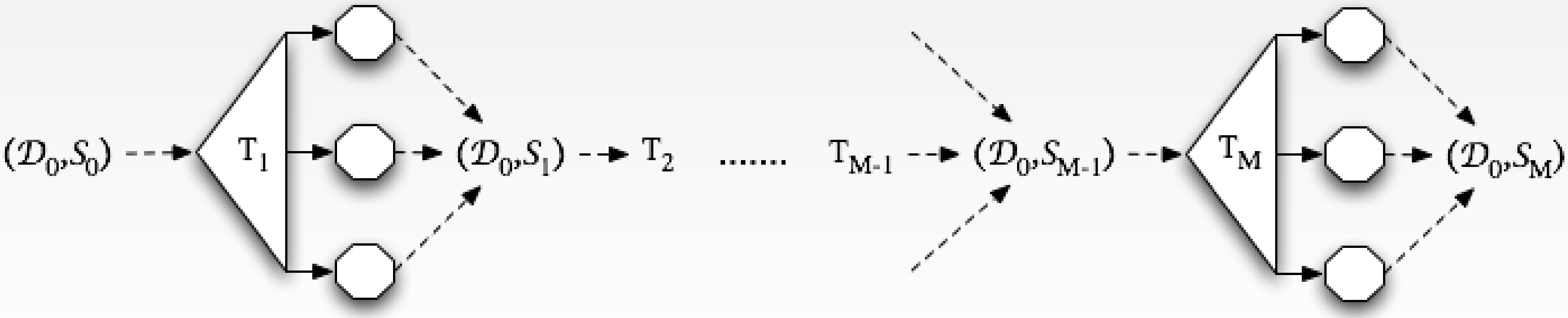
l1
l2
lm
document selection (AND mode)
n
simple scoring function
n
k'
machine learned scoring function
k'
k

- Motivation for improvements
  - efficiency can be improved
    - increased query throughput
    - reduced query response time
  - reduction in hardware costs
  - relevance can be improved
    - more documents can scored by the more accurate ranking system
    - more costly but accurate ranking systems can be afforded



Additive Ranking Systems

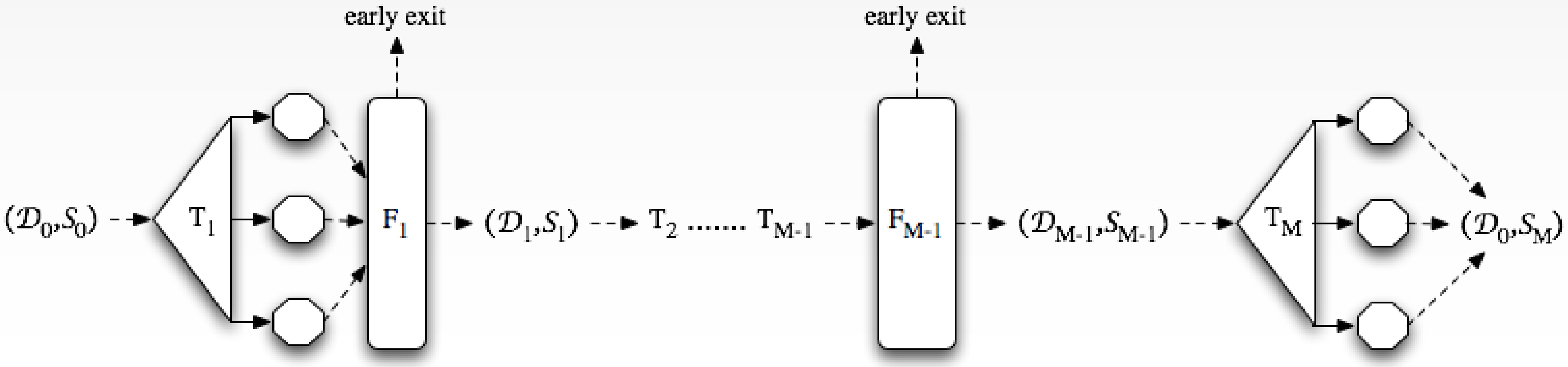
- A chain of machine learned scorers, where each scorer contributes a little to the final score of a document



- Assuming
  - 1000 trees
  - an average tree depth of 10
  - 100 documents scored per query
  - 1000 search nodes
- Expensive
  - 1000*10*100 = around 1 million comparisons per query and per node
  - around 1 billion comparison for the entire search cluster

Early Exit Problem

- Idea: place functions between scorers to predict during scoring whether a document will enter into final top *k* and quit scoring of documents accordingly
- Observations
  - document relevance follows a skewed distribution
  - most users view only the first few results pages



early exit
early exit
$(\mathcal{D}_0,\mathcal{S}_0)$
T_1
F_1
$(\mathcal{D}_1,\mathcal{S}_1)$
T_2 .... T_{M-1}
F_{M-1}
$(\mathcal{D}_{M-1},\mathcal{S}_{M-1})$
T_M
$(\mathcal{D}_0,\mathcal{S}_M)$

- Problem: given a constraint on the run time, minimize the relevance loss due to early exits
- Alternative: given a constraint on the allowed relevance loss, minimize the run time
- Alternative 2: optimize both relevance and run time together as a combined objective

Related Work

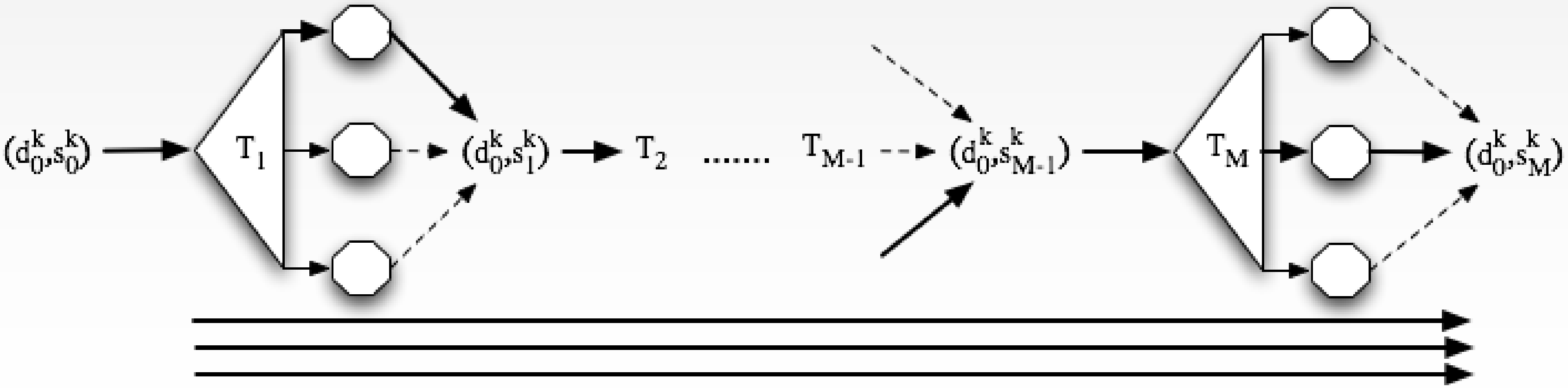
- Additive ensembles
  - SVMs
  - boosting
  - bagging
  - generalized additive models

- Early exit optimizations in vector-space ranking
  - term at a time: Buckley & Lewit (1985); Wong & Lee (1993); Harman & Candela (1990); Persin (1994); Moffat & Zobel (1996); Anh et al. (2001); Anh & Moffat (2006)
  - document at a time: Brown (1995); Turtle & Flood (1995); Strohman et al. (2005)

- Differences from early exit problem in vector-space ranking
  - no prior information available about score contributions
  - expensive early exit algorithms cannot be afforded
  - accumulated scores are not monotonically increasing

Traversal Order

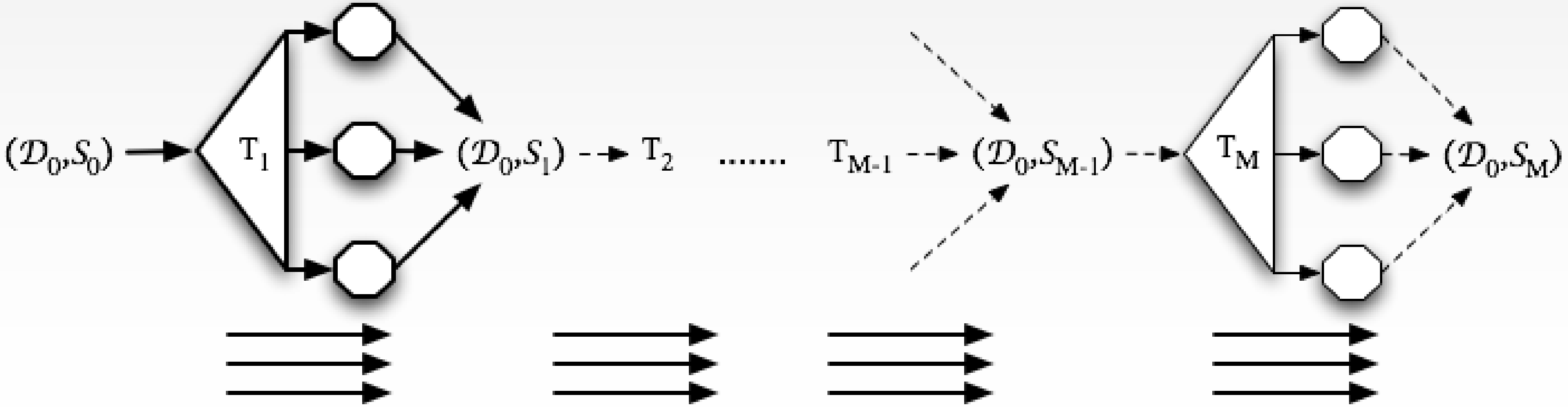
- Document-ordered traversal (DOT)
  - scores are computed one document at a time over all scorers
  - an iteration of the outer loop produces the complete score information for a partial set of documents



- Disadvantages
  - poor branch prediction because a different scorer is used in each inner loop iteration
  - poor cache hit rates in accessing the data about scorers (because of the same reason)

Traversal Order

- Scorer-ordered traversal (SOT)
  - scores are computed one score at a time over all documents
  - an iteration of the outer loop produces the partial score information for the complete set of documents

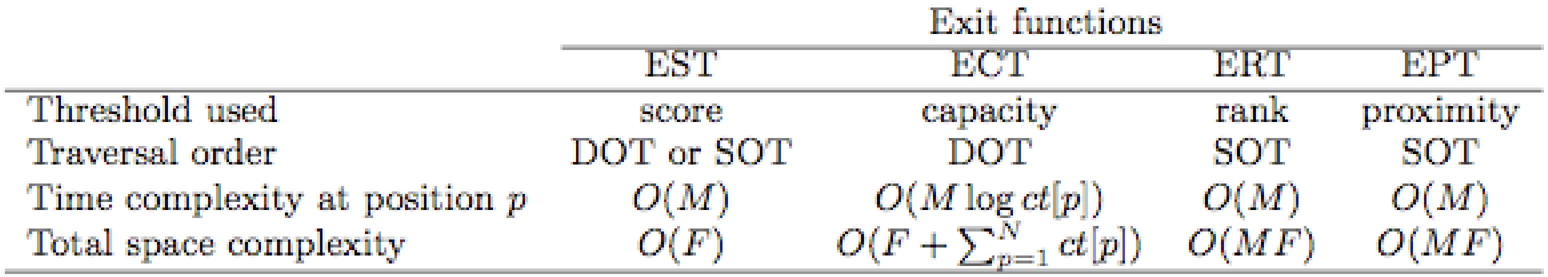


- Disadvantages
  - memory requirement (the feature vectors for all documents need to be kept in memory)
  - poor cache hit rates in accessing features as a different document is used in each inner loop iteration

Early Exit Heuristics

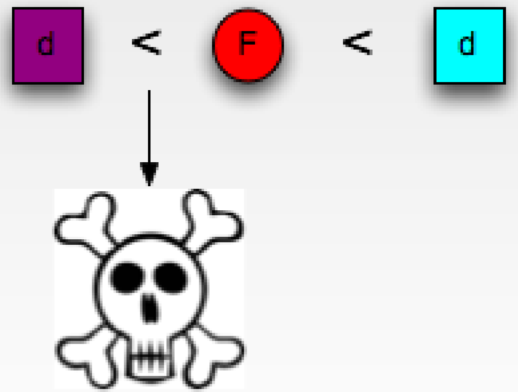
- All early exit heuristics have offline-computed thresholds
- These thresholds determine early exits during the online computation
- Heuristics are named based on the thresholds

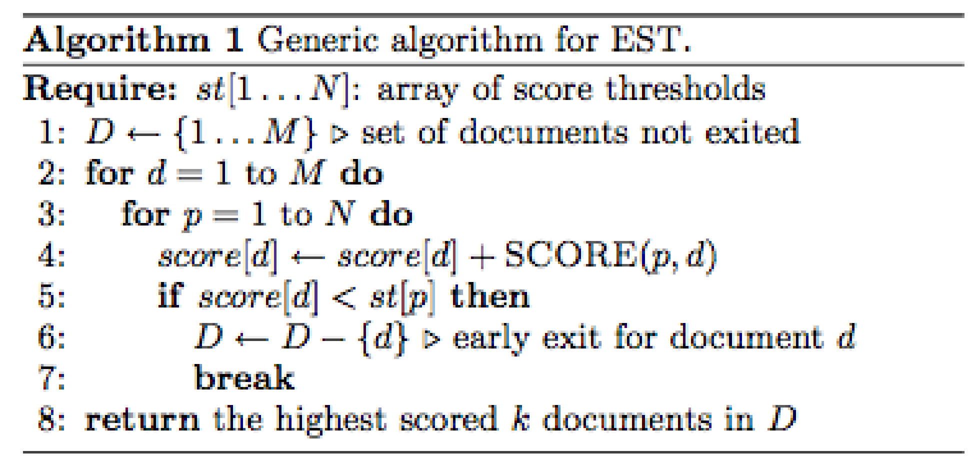
- Heuristics
  - EST: exits with score thresholds
  - ECT: exits with capacity thresholds
  - ERT: exits with rank thresholds
  - EPT: exits with proximity thresholds



	Exit functions			
	EST	ECT	ERT	EPT
Threshold used	score	capacity	rank	proximity
Traversal order	DOT or SOT	DOT	SOT	SOT
Time complexity at position p	$O(M)$	$O(M \log ct[p])$	$O(M)$	$O(M)$
Total space complexity	$O(F)$	$O(F + \sum_{p=1}^{N} ct[p])$	$O(MF)$	$O(MF)$

EST: Exits based on Score Thresholds





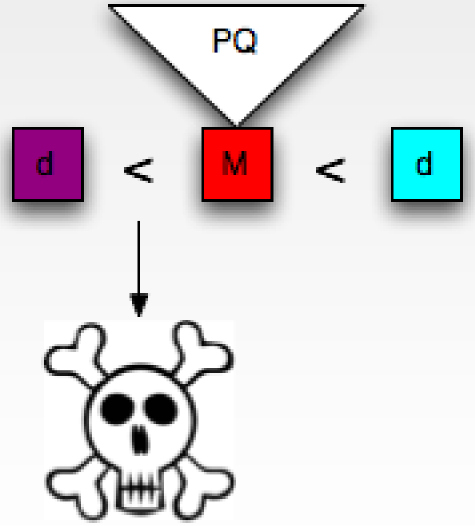
Algorithm 1 Generic algorithm for EST.

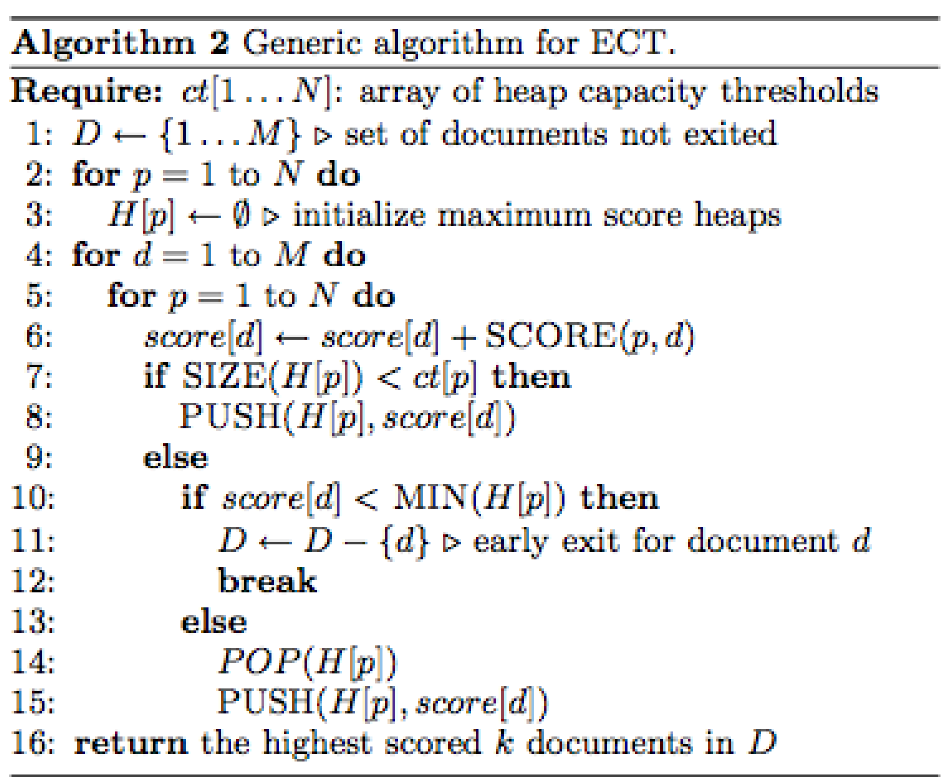
Require: $st[1 \dots N]$: array of score thresholds

```
$D \leftarrow \{1 \dots M\}$ ▷ set of documents not exited
for $d = 1$ to $M$ do
    for $p = 1$ to $N$ do
        $score[d] \leftarrow score[d] + \text{SCORE}(p, d)$
        if $score[d] < st[p]$ then
            $D \leftarrow D - \{d\}$ ▷ early exit for document $d$
            break
return the highest scored $k$ documents in $D$
```

- We early exit a document based on a comparison between the document score accumulated so far and an offline-computed score threshold
- That is, at an exit position, all documents below a certain score threshold *F* are killed
- This heuristic may lead to poor exit decisions because distribution of scores is different for every query

ECT: Exits based on Capacity Thresholds



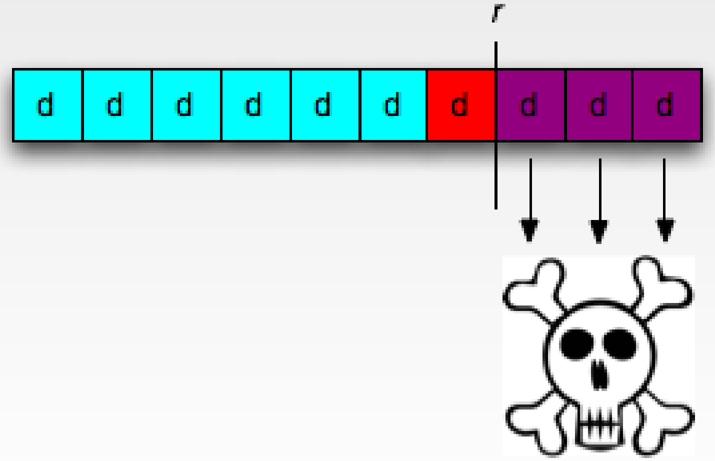


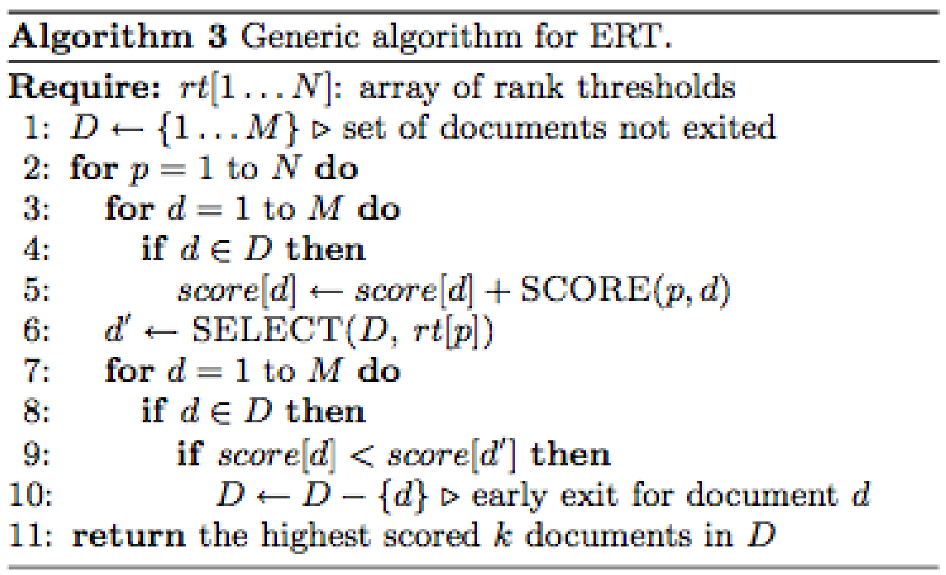
Algorithm 2 Generic algorithm for ECT.

```
Require: $ct[1 \ldots N]$: array of heap capacity thresholds
$D \leftarrow \{1 \ldots M\}$ ▷ set of documents not exited
for $p = 1$ to $N$ do
    $H[p] \leftarrow \emptyset$ ▷ initialize maximum score heaps
for $d = 1$ to $M$ do
    for $p = 1$ to $N$ do
        $score[d] \leftarrow score[d] + \mathrm{SCORE}(p, d)$
        if $\mathrm{SIZE}(H[p]) < ct[p]$ then
            $\mathrm{PUSH}(H[p], score[d])$
        else
            if $score[d] < \mathrm{MIN}(H[p])$ then
                $D \leftarrow D - \{d\}$ ▷ early exit for document $d$
                break
            else
                $POP(H[p])$
                $\mathrm{PUSH}(H[p], score[d])$
return the highest scored $k$ documents in $D$
```

- At every exit position, we maintain a maximum score heap with a certain capacity
- Documents are unconditionally inserted into the heap until it is full
- Afterwards, documents are eliminated via comparisons between their current scores and the minimum score in the heap
- The order in which documents are scored is very important

ERT: Exits based on Rank Thresholds



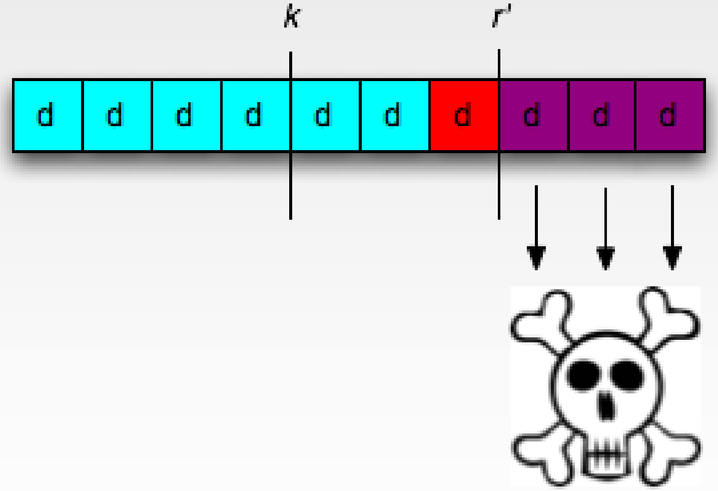


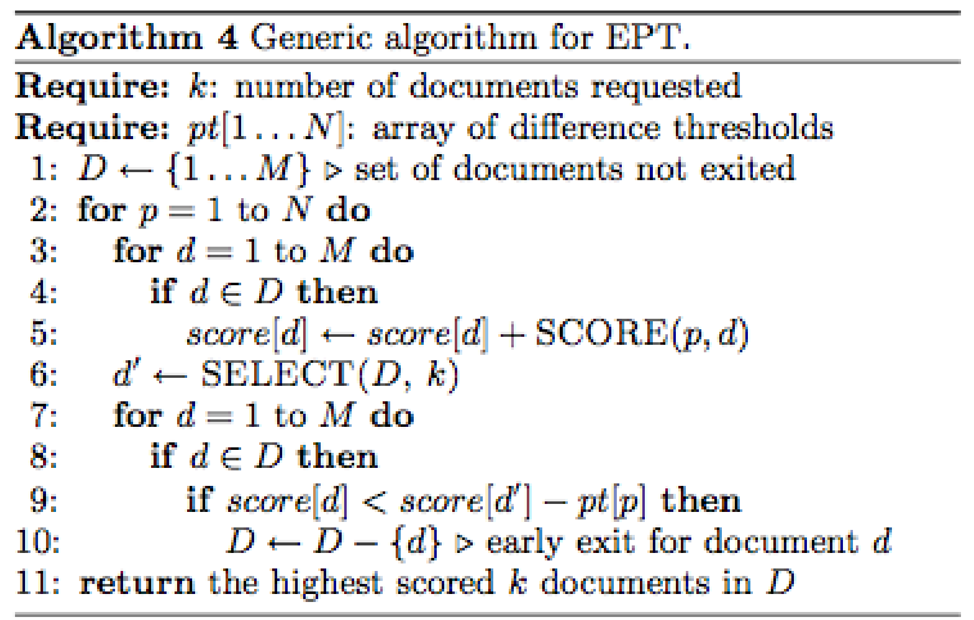
Algorithm 3 Generic algorithm for ERT.

```
Require: $rt[1 \dots N]$: array of rank thresholds
$D \leftarrow \{1 \dots M\}$ ▷ set of documents not exited
for $p = 1$ to $N$ do
    for $d = 1$ to $M$ do
        if $d \in D$ then
            $score[d] \leftarrow score[d] + \mathrm{SCORE}(p, d)$
    $d' \leftarrow \mathrm{SELECT}(D, rt[p])$
    for $d = 1$ to $M$ do
        if $d \in D$ then
            if $score[d] < score[d']$ then
                $D \leftarrow D - \{d\}$ ▷ early exit for document $d$
return the highest scored $k$ documents in $D$
```

- Having the complete ranking after a scorer is quite valuable
- Early exits are performed based on comparisons between the current document ranks and an offline set rank threshold *r*
- The documents with a rank above *r* are allowed for further scoring; the rest are killed
- Linear-time selection algorithm can be used to find the score of the document with rank *r*

EPT: Exits based on Proximity Thresholds





Algorithm 4 Generic algorithm for EPT.

```
Require: $k$: number of documents requested
Require: $pt[1 \ldots N]$: array of difference thresholds
$D \leftarrow \{1 \ldots M\}$ ▷ set of documents not exited
for $p = 1$ to $N$ do
    for $d = 1$ to $M$ do
        if $d \in D$ then
            $score[d] \leftarrow score[d] + \mathrm{SCORE}(p, d)$
    $d' \leftarrow \mathrm{SELECT}(D, k)$
    for $d = 1$ to $M$ do
        if $d \in D$ then
            if $score[d] < score[d'] - pt[p]$ then
                $D \leftarrow D - \{d\}$ ▷ early exit for document $d$
return the highest scored $k$ documents in $D$
```

- We fix the document at rank *k* as the pivot document
- We keep scoring documents that are within a certain score proximity *sp* of the pivot
- Only the documents at first k ranks and those with a score less than *score*[*pivot*]+*sp* are continued to be scored

Experimental Setup

- We use 7400 queries randomly and uniformly sampled from a commercial search engine's query logs.

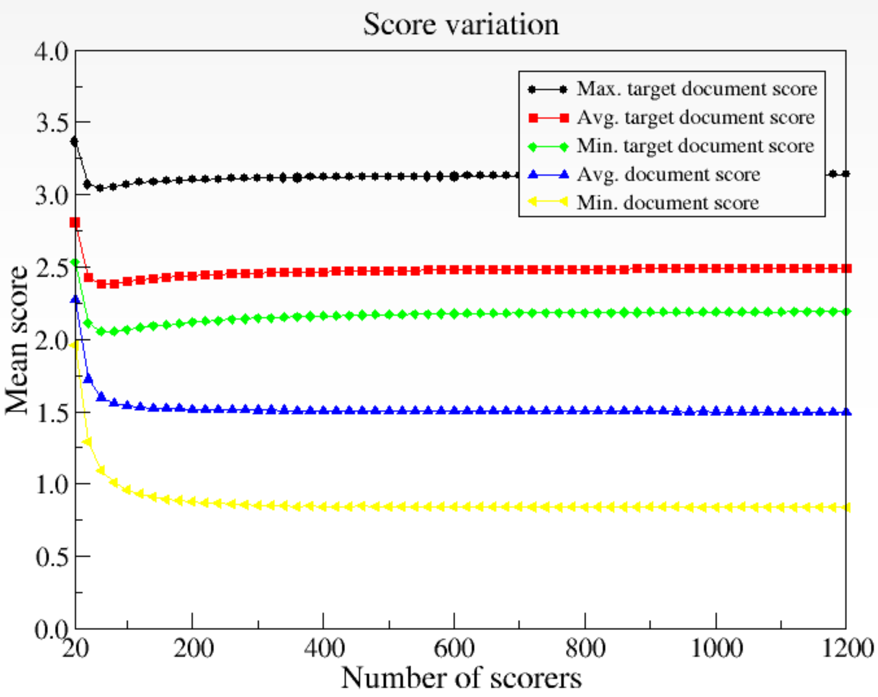
- To form a ground truth, we obtain the top 20 documents computed without any early exits. We call these documents "target documents". Any target document which is eliminated by early exits is said to be missed.

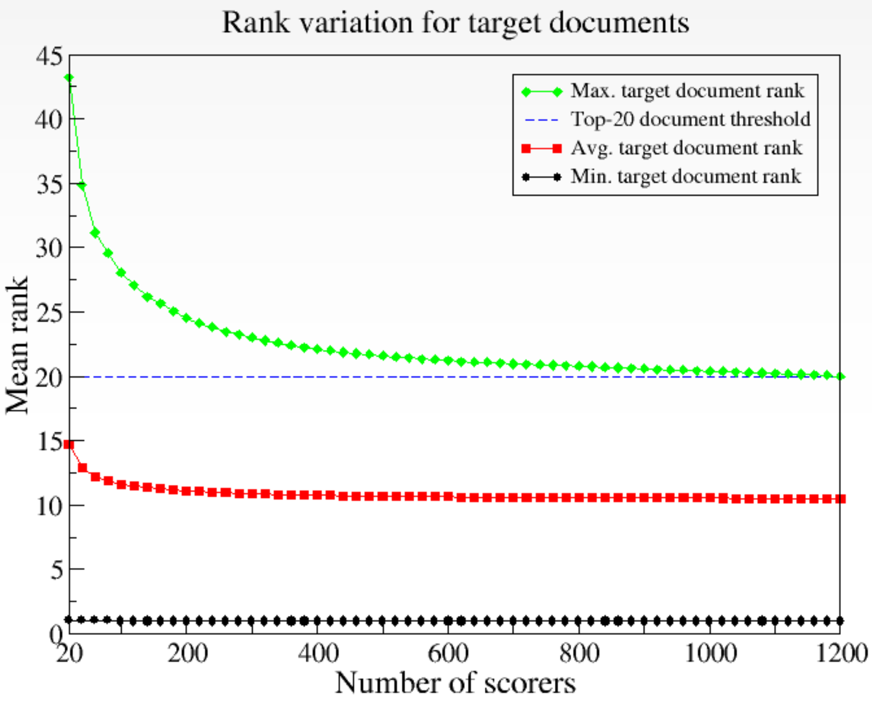
- Documents are evaluated over a machine learned ranking system based on gradient boosted decision trees, composed of 1200 scorers.

- Reported values are averages over all queries.

Behavior of Scores and Ranks

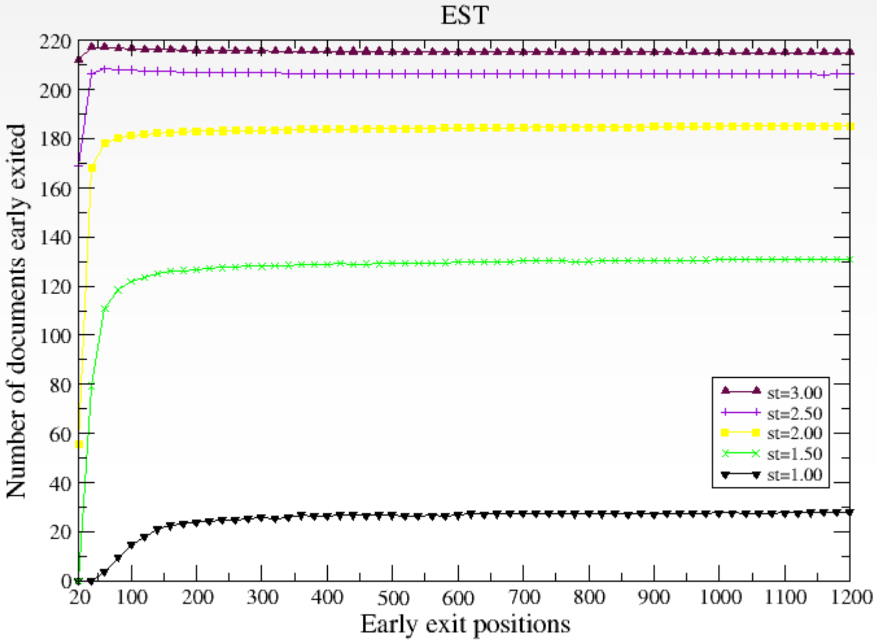
- High score variation at early scorers
- Scores stabilize very quickly as the number of scorers increases
- Average rank of a target document stabilizes faster than the maximum rank



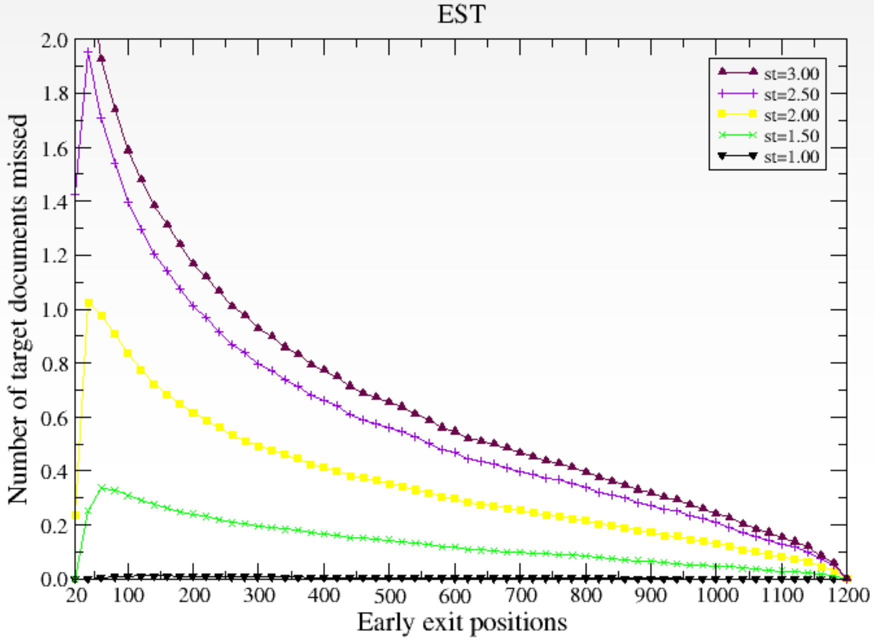


Performance

- Number of early exited documents

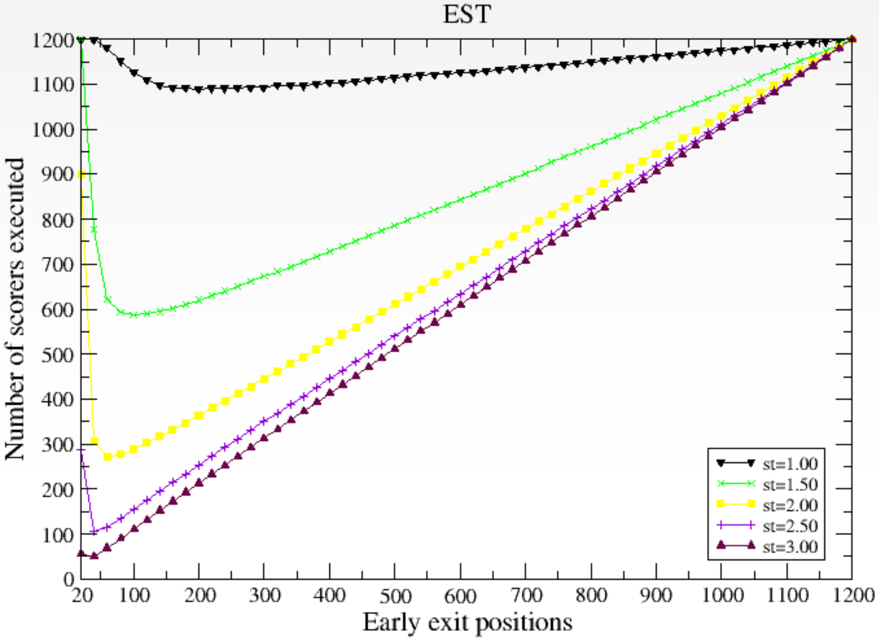


- Number of target documents missed

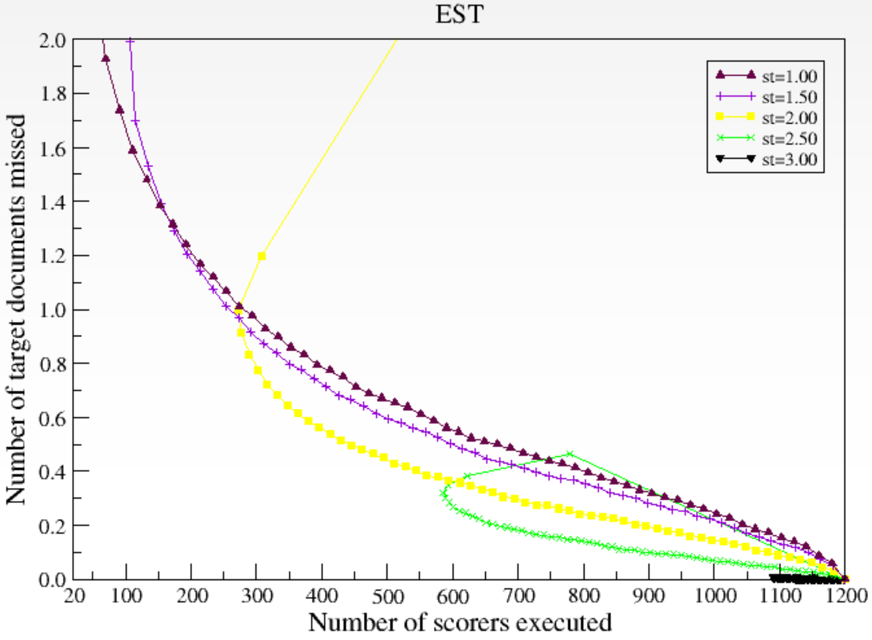


Performance

- Number of scorers executed



- Performance trade-off



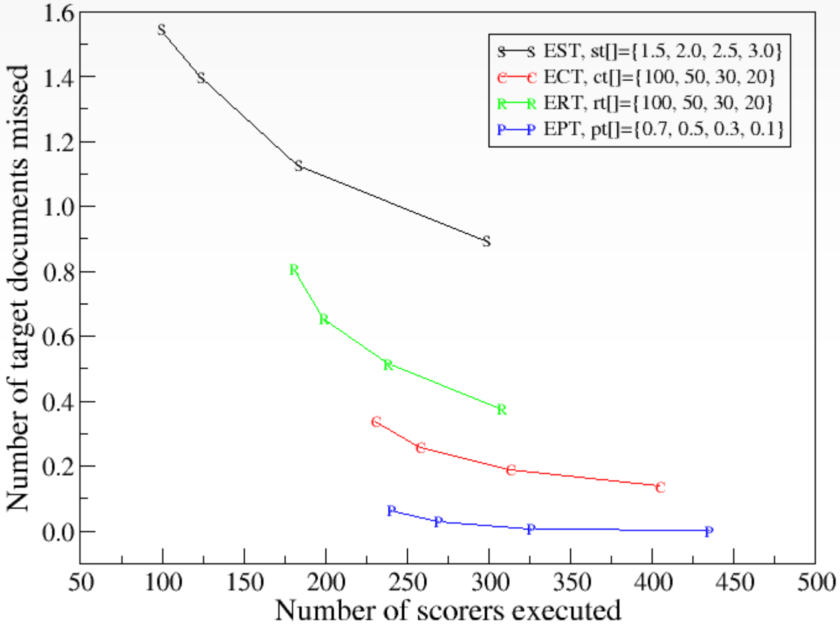
Performance Comparison

- Early exit positions
  - p1[1..4] = {40, 340, 620, 920}
  - p2[1..4] = {40, 160, 400, 740}
  - p3[1..4] = {40, 80, 240, 600}
  - p4[1..4] = {40, 60, 160, 460}

- Thresholds
  - st[] = {1.5, 2.0, 2.5, 3.0}
  - ct[] = {100, 50, 30, 20}
  - rt[] = {100, 50, 30, 20}
  - pt[] = {0.7, 0.5, 0.3, 0.1}

- Performance
  - EPT > ECT > ERT > EST

- EPT leads to almost 4 times speedup without any relevance loss w.r.t. the full score computation



Limitations and Open Problems

- Automate the tuning process for early exit positions and thresholds
- Offline reordering of scorers taking costs of scores into account
- Extend these heuristics to conditional ensembles
- Effect of result cache on the query stream

Any Questions?