Hindawi Publishing Corporation
The Scientific World Journal
Volume 2013, Article ID 960348, 10 pages
http://dx.doi.org/10.1155/2013/960348

Research Article

Knee Point Search Using Cascading Top-*k* Sorting with Minimized Time Complexity

Zheng Wang[1,2] and Shian-Shyong Tseng[3]

[1] *Computer Network Information Center, Chinese Academy of Sciences, Beijing 100190, China*
[2] *China Organizational Name Administration Center, Beijing 100028, China*
[3] *Department of Information Science and Applications, Asia University, Taichung 41354, Taiwan*

Correspondence should be addressed to Zheng Wang; zhengwang09@126.com

Received 20 May 2013; Accepted 20 July 2013

Academic Editors: Z. Cai and Y. Deng

Anomaly detection systems and many other applications are frequently confronted with the problem of finding the largest knee point in the sorted curve for a set of unsorted points. This paper proposes an efficient knee point search algorithm with minimized time complexity using the cascading top-*k* sorting when a priori probability distribution of the knee point is known. First, a top-*k* sort algorithm is proposed based on a quicksort variation. We divide the knee point search problem into multiple steps. And in each step an optimization problem of the selection number *k* is solved, where the objective function is defined as the expected time cost. Because the expected time cost in one step is dependent on that of the afterwards steps, we simplify the optimization problem by minimizing the maximum expected time cost. The posterior probability of the largest knee point distribution and the other parameters are updated before solving the optimization problem in each step. An example of source detection of DNS DoS flooding attacks is provided to illustrate the applications of the proposed algorithm.

1. Introduction

Anomaly detection system and many other applications often rely on finding the largest knee point in the sorted curve to perform clustering, classification, anomaly identification, and so forth [1–6]. Here the largest knee point is targeted because the particular interests lie in finding the cluster of the largest points whose values differ significantly from their lower neighbors in the sorted curve.

Knee point is defined as the point whose value is close to its upper neighbor while far from its lower neighbor in the sorted curve and thereby taken as the boundary of the cluster of upper points. For an unsorted list, it is necessary to sort it to facilitate the knee point search. Due to time and space efficiency considerations, the method of first completely sorting and then searching the sorted list is often not the optimal one. An alternative approach is to perform search on the partially sorted, namely, top-*k*, list, hoping to save the cost of sort. Therefore the top-*k* sort algorithm is introduced to help minimize the time complexity of the knee point search in this paper. There have been many efforts for bounding and evaluating the time and space complexity of sort algorithms [7–12]. These works provide component algorithms for our work. But the problem of knee point search via top-*k* sorting has not been addressed by any of the previous works. We present in this paper a knee point search algorithm using top-*k* sorting with minimized time complexity.

This paper is organized as follows: some basic concepts and definitions on knee point search and top-*k* sorting are presented in Section 2; Section 3 will design a knee point search algorithm, including basic idea, top-*k* sort algorithm, time complexity, parameter updating, cascading top-*k* sorting with minimized time complexity, the knee point search algorithm, and the solution of the optimization problem; Section 4 will introduce source detection of DNS DoS flooding attacks as an application example of the proposed algorithm; Section 5 will conclude this paper.

2. Knee Point Search and Selection Sort

Assume there are n points $a_1, a_2, \ldots, a_n$ whose values are $q_1, q_2, \ldots, q_n$. Let $a_1, a_2, \ldots, a_n$ be sorted by their values and

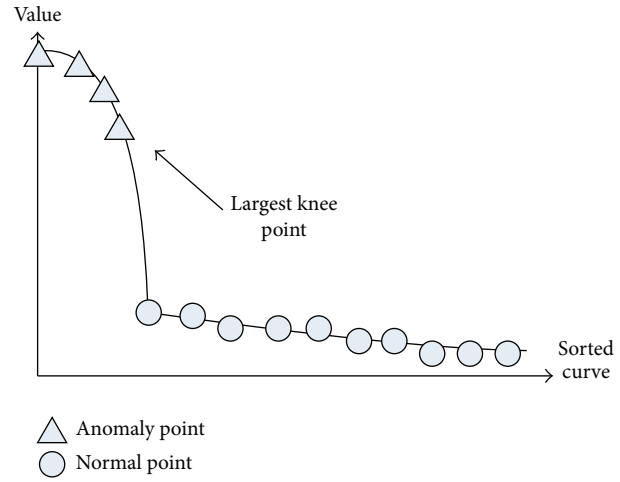

Figure 1: Sorted graph for first knee point search.

let their differential values be $\delta_1, \delta_2, \ldots, \delta_n$, where $q_1 \geq q_2 \geq \cdots \geq q_n$, and

$$\delta_i = \begin{cases} \delta_1 & i = 1 \\ q_{i-1} - q_i & i = 2, 3, \ldots, n. \end{cases} \quad (1)$$

As illustrated in Figure 1, there is usually a notable gap of value between points on the upper left side and those on the lower right side of the sorted curve. We define a knee point as the one whose neighboring differential values differ significantly.

Definition 1. Point a_i, $i = 1, 2, \ldots, n-1$, is a knee point if it satisfies

$$\frac{\delta_{i+1}}{\delta_i} > \theta, \quad (2)$$

where θ is the threshold, whose value ranges from 10 to 50 in the practice of anomaly detection.

Note that there may be more than one knee point for a list, and the goal of the algorithm is to find the largest one in the sorted curve. For an unsorted list, we should first partially sort the list to find the sorted top-k list and then search the sorted top-k list for the largest knee point.

Definition 2. A top-k sort problem of selection number k for an unsorted list L is a problem that finds k largest elements of L sorted in descending order.

Apparently, total sort is often not optimal for the problem as knee point search may be successful on a partially sorted top-k list if it contains the largest knee point. Therefore it is preferable to selectively sort first using the top-k sort algorithm and then search in each step. The procedure may go through many recursive steps until finding the largest knee point for the search may fail in the previous steps. There is a tradeoff between the time cost and expected hit probability of knee point search in each step, both determined by the selection number and both contributing to the expected overall time cost. In this paper, the optimization problem of the selection number is solved by minimizing the maximum expected time cost.

3. The Knee Point Search Algorithm

3.1. Basic Idea. The knee point search algorithm is based on cascading top-k sorting. In each step, top-k sorting segments the list left to be searched. The optimal selection number is determined by minimizing the expected time cost on the list left. If the search successfully finds the knee point in the sorted top-k list, the algorithm ends there. Otherwise, the residual list excluding top-k requires further checking. It becomes the objective list for the next step and the function of the expected time cost using top-k sorting should also be updated according to the a priori knowledge that the search fails in the previous step. Thus the new top-k sort problem, likewise, holds for the next step. The algorithm runs in this way recursively until the knee point is found. There are two cases that bring the algorithm to the end.

(1) The knee point is found in the sorted top-k list in a step.

(2) The optimal selection number in one step equals the length of the objective list. This means the optimal option is total sorting. Therefore the knee point is certain to be found in the completely sorted list.

3.2. Top-k Sort Algorithm. We design a quicksort variation as the top-k sort algorithm. Quicksort is a very efficient sort algorithm invented by Hoare [7]. Quicksort has two phases: the partition phase and the sort phase, which makes it a good example of the divide and conquer strategy for solving problems.

Top-k sorting only aims at treating the largest k elements, and thus it can be facilitated by the divide and conquer strategy. The intermediate results of quicksort, namely, the pivot positions, can be leveraged to possibly cut off one of the smaller problems divided from the bigger problem and to be conquered. For the strategy to be effective, the partition phase runs recursively only for the lower part if the pivot falls below position k, because there is no need to sort the upper part, which only consists of elements larger than top-k. This is the major distinction from the original quicksort algorithm and brings sorting efficiency.

At the same time, the pivots located after position d_i (the optimal selection number) at step i in Section 3.6 are potentially useful for the afterwards steps, while they are actually not helpful for the inner top-d_i sorting. Therefore we record those pivots via a stack. A stack is a data structure featured by last in, first out (LIFO). Recalling that the recursive partitions with their pivots after position k produce their pivots in a sequential descending position order, we push these pivots into the stack resulting in a stack of pivots ordered by their positions. At the afterwards step $i + 1$, if the optimal selection number d_{i+1} is larger than the position of the pivot at the top of the stack, the pivot is popped from the stack used for an inner pivot of top-d_{i+1}. Since this pivot is no longer needed for the afterwards steps, it should not be

maintained in the stack. When the stack is empty or the pivot at the top of the stack (so do all of the other pivots) is located after the selection number, the partition has to run by itself to find a pivot without the help of the pivot stack.

The top-k sort algorithm, namely, QuickSortTopK, can be expressed as in Algorithm 1.

The input of QuickSortTopK is the objective unsorted list L indexed from FirstIndex to LastIndex. For all steps, LastIndex is fixed at n, whereas FirstIndex is progressively increased to exclude the sorted part of L in all previous steps. The output includes the sorted top-k elements of L indexed from FirstIndex to LastIndex and the stack S containing all pivots falling after position k obtained in all previous steps.

The termination condition of the recursion is checked in Line 1 of the algorithm. If the stack is nonempty and the top element of the stack falls into the objective range (see Line 2), the top element is used as the pivot for the partition (see Line 3). Otherwise, the pivot is obtained by a partition (see Line 6). Once the pivot is presented, different recursive steps are to be taken depending on the position of the pivot. If the pivot falls after position k, it should be pushed into the stack and then run further sorting on the original list subtracting the pivot, hoping to help afterwards steps (see Lines 9, 11). If the pivot is located exactly at position k, the pivot itself is the last element of the output and thereafter only top-k-1 sorting on the original list subtracting the pivot is needed (see Line 15). If the pivot is located prior to position k, both the upper and lower parts should be treated. The action on the upper part is equivalent to Quicksort, while the action on the lower part is actually the recursive running of QuickSortTopK with diminished selection number k and the shrinking objective list (see Lines 18, 20).

3.3. Time Complexity. Let totally sorting of a list of length n require $S(n)$ time. Calculated by the number of comparisons, the average time complexity of $S(n)$ is $O(n\log(n))$ following some efficient algorithms, for example, Quicksort [7].

Let top-k sorting of selection number k require $C(k,n)$ time, where n is the length of the list. The QuickSortTopK algorithm requires an expected time of $O(n + k\log k)$. So $C(k,n)$ equals $O(n + k\log k)$.

Let the time complexity of finding the knee point in the sorted list of length n be $D(n)$. Recalling (2), $D(n)$ takes $O(n)$.

3.4. Parameter Updating. For a list of length n, the algorithm divides the overall procedure into $m+1$ steps by a sequence of selection numbers, $d_1, d_2, \ldots, d_m$, $1 \le d_1 < d_2 < \cdots < d_m \le n$. Additionally, to facilitate the formulation, we let $d_0 = 0$ and $d_m + 1 = n$. Let the length of the objective list in step i, $i = 1, 2, \ldots, m+1$ be n_i; we have

$$n_i = n - d_{i-1}, \quad i = 1, 2, \ldots, m+1. \tag{3}$$

In the first step, $n_i = n$. Let top-k_i sorting for the objective list of length n_i be performed in step i, $i = 1, 2, \ldots, m+1$, and we have

$$k_i = d_i - d_{i-1}, \quad i = 1, 2, \ldots, m+1. \tag{4}$$

Particularly, $k_i = n_i$ in step $m+1$. And thus top-k_i sorting for the objective list of length n_i is actually total sorting of the objective list. If the search of the knee point is successful in step i for the sorted top-k_i list, the algorithm ends at step i. Otherwise, the algorithm continues with the next step. The algorithm lasts until step $m+1$ if the search misses in all of the previous steps during the progressive search. Since step $m+1$ takes no further selection of the objective list, the algorithm finishes in it.

Let A be the position variable of the knee point and $A = 1, 2, \ldots, n$. Let $p(i)$ represent the probability that $A = i$, $i = 1, 2, \ldots, n$, and thus $\sum_{i=1}^{n} p(i) = 1$. The value of $p(i)$ is assumed to be known at the beginning of the algorithm. Let A_i be the position variable of the knee point in step i and $A_i = d_{i-1} + 1, d_{i-1} + 2, \ldots, n$. Let $p_i(j)$ represent the probability that $A_i = j$, $j = d_{i-1} + 1, d_{i-1} + 2, \ldots, n$, $i = 1, 2, \ldots, m+1$, and thus $\sum_{j=d_{i-1}+1}^{n} p_i(j) = 1$.

Lemma 3. *The probability distribution of the knee point in step i $p_i(j)$ can be written as*

$$p_i(j) = \frac{p(j)}{\sum_{k=d_{i-1}+1}^{n} p(k)}, \quad j = d_{i-1}+1,\ d_{i-1}+2, \ldots, n, \quad i = 1, 2, \ldots, m+1. \tag{5}$$

Proof. At the first step, all knowledge about the probability distribution of the knee point is only given by $p(i)$. But the search in the afterward steps should make use of the posterior distribution of the knee point for it is confirmed not to exist prior to the selection number in the previous steps; for example, when the algorithm comes to step i, the knee point is already checked to be not present in the top-d_{i-1} list, $i = 2, 3, \ldots, m+1$. Therefore $p_i(j)$ should be updated in step i as

$$\begin{aligned} p_i(j) &= P(A_i = j) \\ &= P\left(A = j \mid A \in \{d_{i-1}+1, d_{i-1}+2, \ldots, n\}\right) \\ &= \frac{P\left(A = j, j \in \{d_{i-1}+1, d_{i-1}+2, \ldots, n\}\right)}{P\left(A \in \{d_{i-1}+1, d_{i-1}+2, \ldots, n\}\right)} \\ &= \frac{p(j)}{\sum_{k=d_{i-1}+1}^{n} p(k)} \\ & j = d_{i-1}+1,\ d_{i-1}+2, \ldots, n, \quad i = 1, 2, \ldots, m+1. \end{aligned} \tag{6}$$

□

Let the hit probability of search in step i be H_i.

Lemma 4. *The hit probability of search in step i H_i can be written as*

$$H_i(d_i) = \sum_{j=d_{i-1}+1}^{d_i} p_i(j), \quad i = 1, 2, \ldots, m+1. \tag{7}$$

Proof. For the selection number d_i in step i, the search for the knee point is successful if and only if the knee point falls

```
Input:
L: an unsorted list of length n
FirstIndex: the first index of L to be sorted
LastIndex: the last index of L to be sorted
k: top-k elements of L are to be sorted
Output:
L_k: Sorted top-k elements of L ranging from FirstIndex to LastIndex
S: a stack of pivot positons useful for the afterwards selection sorts
(1)  if FirstIndex < LastIndex then
(2)      if !isempty(S) & top(S) < LastIndex then        /* the stack is available */
(3)          pivotpos = pop(S)        /* use the top element of the stack as the pivot */
(4)      else
(5)          /* Partition without using pivots from the stack */
(6)          pivotpos=Partition(L, FirstIndex, LastIndex)
(7)      if pivotpos– FirstIndex +1> k then    /* The pivot falls after position k */
(8)          /* The pivot may be useful for the afterwards steps */
(9)          push(pivotpos, S)
(10)         /* QuickSort for top-k in the upper part */
(11)           QuickSortTopK(L, FirstIndex, pivotpos –1, k)
(12)     /* The pivot is located exactly at position k */
(13)     elseif(pivotpos– FirstIndex +1==k)
(14)         /* QuickSort for top-k – 1 in the upper part */
(15)         QuickSortTopK(L, FirstIndex, pivotpos –1, k – 1)
(16)         else       /* The pivot falls prior to position k */
(17)             /* QuickSort the upper part */
(18)             QuickSortTopK(L, FirstIndex, pivotpos –1, pivotpos– FirstIndex)
(19)             /* QuickSort for the residual elements in the lower part */
(20)             QuickSortTopK(L, pivotpos +1, FirstIndex, k-pivotpos+ FirstIndex –1)
```

Algorithm 1: QuickSortTopK(L, FirstIndex, LastIndex, k).

into the interval of position among $d_{i-1}+1$ and d_i. According to Lemma 3, the probability distribution of the knee point at step i $p_i(j)$ should be updated as (5). Therefore we have

$$H_i(d_i) = \sum_{j=d_{i-1}+1}^{d_i} P(A_i = j) = \sum_{j=d_{i-1}+1}^{d_i} p_i(j), \quad i = 1, 2, \ldots, m+1. \tag{8}$$

□

3.5. Cascading Top-k Sorting with Minimized Time Complexity

Lemma 5. *Let the expected overall computational time cost in step i be tc_i; tc_i yields*

$$tc_i = C(d_i - d_{i-1}, n - d_{i-1}) + (1 - H_i(d_i))\, tc_{i+1}, \quad i = 1, 2, \ldots, m. \tag{9}$$

Proof. When the search succeeds in step i, tc_i comes only from top-k_i sorting which requires $C(k_i, n_i)$ time and the search in the top-k_i sorted list which requires $D(n)$. However, recalling Section 3.3, the time complexity of $D(n)$ is negligible compared to that of $C(k_i, n_i)$, so the summation of them can be approximated by $C(k_i, n_i)$. For the failure of search in step i, the residual list of length $n_i - d_i$ has to be further checked. Thus the overall computation time cost consists of top-k sort and search in the remaining list. Note that the computation time cost of sort and search in the remaining list is no other than tc_{i+1}. Hence tc_i yields

$$\begin{aligned}\mathrm{tc}_i &= H_i(d_i)\, C(k_i, n_i) + (1 - H_i(d_i))\,(C(k_i, n_i) + \mathrm{tc}_{i+1}) \\ &= C(k_i, n_i) + (1 - H_i(d_i))\,\mathrm{tc}_{i+1}, \quad i = 1, 2, \ldots, m.\end{aligned} \tag{10}$$

Plugging (3) and (4) into (10), we get (9). □

Lemma 5 tells us that the expected computational cost tc_1 can be calculated iteratively following (7), until reaching tc_{m+1} where there is no selection for step $m+1$. Thus tc_{m+1} only consists of the time cost of total sorting of the list of length n_i and the search in it. As total sorting takes $S(n_i)$ and search takes $D(n_i)$, thus we have

$$\mathrm{tc}_{m+1} = S(n_{m+1}) + D(n_{m+1}). \tag{11}$$

Let $N(a, b)$ ($a \le b$ and a, b are integers) denote the set of integers $\{i \mid a \le i \le b \text{ and } i \text{ is integer}\}$. In every step i, the algorithm calculates the current probability distribution of the knee point which determines the hit probability H_i and

chooses d_i to be the solution of the following optimization problem:

$$\begin{aligned} &\text{Min}: \quad \text{tc}_i \\ &\text{s.t.} \quad\quad d_i \in N\left(d_{i-1}+1, n\right). \end{aligned} \tag{12}$$

We see in (12) that for any fixed d_i the minimum of tc_i is determined by tc_{i+1} under the optimal selection of d_{i+1} in the next step, $d_i \in N(d_{i-1}+1, n)$. And the optimal tc_{i+1} for any fixed d_{i+1} is also determined by the tc_{i+2} under the optimal selection of d_{i+2} and so on. This kind of iterative dependency finally extends to the last step which has no further selection. So tc_i is the function of $d_i, d_{i+1}, \ldots$. The variation of any choice of selection in any number of steps makes the search space of optimization very huge, especially for the initial steps. Therefore it is not practical to evaluate all possibilities of selections in all of the afterwards steps when solving the optimization problem in step i. Thus it is necessary to constrain the variable of the objective function tc_i in (12) as mere d_i.

Theorem 6. *The upper bound of the minimum of tc_i for a fixed d_i can be written as substituting tc_{i+1} in (9) by $S(n_{i+1})+D(n_{i+1})$, such that*

$$\begin{aligned} &\min_{d_{i+1}\in N(d_i+1,n), d_{i+2}\in N(d_{i+1}+1,n),\ldots} tc_i \\ &\quad \le C\left(d_i - d_{i-1}, n - d_{i-1}\right) \\ &\qquad + \left(1 - H_i\left(d_i\right)\right)\left(S\left(n-d_i\right) + D\left(n-d_i\right)\right), \\ &\qquad\qquad i = 1, 2, \ldots, m. \end{aligned} \tag{13}$$

Proof. We only need to prove that

$$\min_{d_{i+1}\in N(d_i+1,n), d_{i+2}\in N(d_{i+1}+1,n),\ldots} \text{tc}_{i+1} \le S\left(n-d_i\right) + D\left(n-d_i\right). \tag{14}$$

As total sort of the list of length n_{i+1} can be viewed as performing top-n_{i+1} sorting, their time costs are both $S(n_{i+1}) + D(n_{i+1})$. This is equivalent to the case when $d_{i+1} = n$. Because the algorithm ends with the total sorting and there are no afterwards steps, we have no definitions for d_{i+1}, $d_{i+2}, \ldots$. Thus we have

$$\text{tc}_{i+1}\big|_{d_{i+1}=n \text{ (hence } d_{i+2}, d_{i+3\ldots} \text{ has no definition)}} = S\left(n_{i+1}\right) + D\left(n_{i+1}\right). \tag{15}$$

Hence

$$\begin{aligned} &\min_{d_{i+1}\in N(d_i+1,n), d_{i+2}\in N(d_{i+1}+1,n),\ldots} \text{tc}_{i+1} \\ &\quad \le \text{tc}_{i+1}\big|_{d_{i+1}=n \text{ (hence } d_{i+2}, d_{i+3,\ldots} \text{has no definition)}}. \end{aligned} \tag{16}$$

Plugging (3) into (15) and then (15) into (16), we have (14), and thereby (13) is proved. □

Theorem 6 manifests that, for a fixed d_i, tc_i is definitely bounded by the time cost of total sort of the residual list of length $n_i + 1$ plus that of search in it. The optimization problem in step i described by (13) can be isolated from all of the possible selections of the afterwards steps and becomes a function of mere d_i. This minmax technique brings convenience to our analysis, such that (13) can be simplified as

$$\begin{aligned} &\text{Min}: \quad C\left(d_i - d_{i-1}, n - d_{i-1}\right) \\ &\qquad + \left(1 - H_i\left(d_i\right)\right)\left(S\left(n-d_i\right) + D\left(n-d_i\right)\right) \\ &\text{s.t.} \quad\quad d_i \in N\left(d_{i-1}+1, n\right). \end{aligned} \tag{17}$$

Plugging (5) into (17), we have

$$\begin{aligned} &\text{Min}: \quad C\left(d_i - d_{i-1}, n - d_{i-1}\right) \\ &\qquad + \left(1 - \sum_{j=d_{i-1}+1}^{d_i} p_i\left(j\right)\right)\left(S\left(n-d_i\right) + D\left(n-d_i\right)\right) \\ &\text{s.t.} \quad\quad d_i \in N\left(d_{i-1}+1, n\right). \end{aligned} \tag{18}$$

Solving (14), we can obtain the optimal d_i in step i.

3.6. The Knee Point Search Algorithm. The knee point search algorithm runs iteratively using cascading top-k sorting. When the optimal selection number d_i is determined at step i, top-k sorting can be done via running QuickSortTopK $(L, d_{i-1}+1, n, d_i - d_{i-1})$. Specifically, the first step starts with QuickSortTopK $(L, 1, n, d_1)$.

The procedure can be described as follows.

Step 1.

(1) According to (5), the probability distribution of the knee point for the optimization problem is initialized as follows

$$p_1\left(j\right) = p\left(j\right), \quad j = 1, 2, \ldots, n. \tag{19}$$

(2) The optimal selection number d_1 is obtained by solving the following optimization problem as (18):

$$\begin{aligned} &\text{Min}: \quad C\left(d_1, n\right) + \left(1 - \sum_{j=1}^{d_1} p\left(j\right)\right) \\ &\qquad \times \left(S\left(n-d_1\right) + D\left(n-d_1\right)\right) \\ &\text{s.t.} \quad\quad d_1 \in N\left(1, n\right). \end{aligned} \tag{20}$$

(3) Perform *top-d_1* sorting on the list of length n.

(4) Search for the knee point on the sorted top-d_1 list. If successful or $d_1 = n$, the algorithm ends. Otherwise, go to Step 2.

Step 2.

(1) The probability distribution of the knee point for the optimization problem is updated as follows:

$$p_2\left(j\right) = \frac{p\left(j\right)}{\sum_{k=d_1+1}^{n} p\left(k\right)}, \quad j = d_1 + 1,\ d_1 + 2, \ldots, n. \tag{21}$$

(2) d_2 is derived as the solution of the following optimization problem:

$$\begin{aligned} \text{Min}: \quad & C(d_2 - d_1, n - d_1) + \left(1 - \sum_{j=d_1+1}^{d_2} p_2(j)\right) \\ & \times \left(S(n - d_2) + D(n - d_2)\right) \\ \text{s.t.} \quad & d_2 \in N(d_1 + 1, n). \end{aligned} \tag{22}$$

Note that d_1 is inherited from Step 1.

(3) Perform top-$(d_2 - d_1)$ selection sort on the list of length $n - d_1$ subtracting top-d_1 elements in Step 1.

(4) Search for the knee point on the sorted top-$(d_2 - d_1)$ list. If successful or $d_2 = n$, the algorithm ends. Otherwise, go to Step 2.

Step i.

(1) The probability distribution of the knee point for the optimization problem is updated according to (5).

(2) d_i is obtained as the solution of the optimization problem in (18), where d_{i-1} is known from step $i - 1$.

(3) Perform top-$(d_i - d_{i-1})$ selection sort on the list of length $n - d_{i-1}$ subtracting top-d_{i-1} elements already sorted in the previous steps.

(4) Search for the knee point on the sorted top-$(d_i - d_{i-1})$ list. If successful or $d_i = n$, the algorithm ends. Otherwise, go to step $i + 1$.

The knee point search algorithm can be summarized as in Algorithm 2.

The knee point search algorithm can also be expressed recursively.

For each recursive step, we have an unsorted list L of length n and the probability distribution of the knee point in the sorted list of length n: P. Thus we can modify the optimization problem in (18) as follows:

$$\begin{aligned} \text{Min}: \quad & C(d, n) + \left(1 - \sum_{j=1}^{d} p(j)\right)(S(n - d) + D(n - d)) \\ \text{s.t.} \quad & d \in N(1, n). \end{aligned} \tag{23}$$

Solving (14), we can obtain the optimal d in each recursive step.

When the knee point search fails after top-k sorting in each recursive step, the algorithm has to go to the next recursive step. First, we need to update the probability distribution of the knee point as well as the residual unsorted list as two parameters for the recursive function. According to Lemma 3, the update of the probability distribution of the knee point yields

$$p(j) \longleftarrow \frac{p(j)}{\sum_{k=d+1}^{n} p(k)}, \quad j = d + 1,\ d + 2, \ldots, n. \tag{24}$$

In each recursive step, top-k sorting can be done via running QuickSortTopK(L, 1, length(L), d), where L is the objective list in the current step and *length* (L) denotes the length of L.

A recursive version of knee point search algorithm can be summarized as in Algorithm 3

3.7. The Solution of the Optimization Problem. In this section, we will assume two forms of the probability distribution of the knee point and discuss the solutions of (18) under these presumptions.

(1) Uniform Distribution. $p(i)$ is equal for all $i \in N(1, n)$, and thus $p(i) = 1/n$. Plugging $p(i)$ into (23), we have

$$\begin{aligned} \text{Min}: \quad & C(d, n) + \left(1 - \frac{d}{n}\right)(S(n - d) + D(n - d)) \\ \text{s.t.} \quad & d \in N(1, n). \end{aligned} \tag{25}$$

Let

$$f(d) = C(d, n) + \left(1 - \frac{d}{n}\right)(S(n - d) + D(n - d)). \tag{26}$$

Although (26) is a discrete function, we still utilize the method of derivation to find the extremum, which can be only applied to the continuous and derivable function. Here we treat the discrete variable d as continuous ones, and thereafter (26) turns into a continuous and derivable function. This is a rational approximation of the problem, which facilitates our analysis and solving. The final solution should be the round-off of d obtained by solving the continuous function.

For simplicity, we let

$$\begin{aligned} C(k, n) &= c_1(n + k \log k) \\ S(n) &= c_2(n \log(n)) \\ D(n) &= c_3 n, \end{aligned} \tag{27}$$

where c_1, c_2, and c_3 are all constants.

By choosing d such that $df(d)/d = 0$, we have the optimal d which satisfies

$$c_1(1 + \log d)\, n = (n - d)\left[2c_2 \log(n - d) + 2c_3 + c_2\right]. \tag{28}$$

For large n and d, we have the approximation of (28) as

$$c_1 n \log d = 2c_2 (n - d) \log(n - d). \tag{29}$$

To get the explicit mathematical expression of the solution of the nonlinearity equation (29), we used a heuristic approach to simplify the problem. We assume that d is a proportion function of n, such that

$$d = pn, \quad 0 < p < 1, \tag{30}$$

where p decides the optimum of d.

Thus (30) yields

$$c_1 n \log(pn) = 2c_2 (1 - p)\, n \log\left[(1 - p)\, n\right], \tag{31}$$

Input: Unsorted list L of length n, probability distribution of the knee point in the sorted list of length n: P
Output: The knee point e or the non-existence of it (e = null).
(1) /* initialize the parameters for the optimization problem */
(2) $P_1 \leftarrow P, L_1 \leftarrow L$
(3) $i \leftarrow 1$
(4) **while** (true)
(5) solve the optimization problem of list L_i in (18) to get d_i
(6) **if** $d_i = n_i$ **then** /* total sorting is the optimal option */
(7) totally sort list L_i
(8) /* successful if the knee point exists */
(9) Search the sorted list for the knee point e
(10) **break** /* the algorithm is completed */
(11) **else** /* top-$(d_i - d_{i-1})$ sort is the optimal option */
(12) QuickSortTopK(L, $d_{i-1} + 1$, n, d_i–d_{i-1})
(13) Search the sorted list for the knee point e
(14) **if** e is found **then**
(15) **break**
(16) **else** /* Update the parameters for the next round of optimization problem */
(17) update P_{i+1} according to (5) /* the posterior probability */
(18) update L_{i+1} by exluding the top-$(d_i - d_{i-1})$ item from list L_i
(19) $i \leftarrow i + 1$
(20) **return** e

Algorithm 2: FindKnee(L, P).

Input: Unsorted list L of length n, probability distribution of the knee point in the sorted list of length n: P
Output: The knee point e or the non-existence of it (e = null).
(1) solve the optimization problem of list L in (20) to get d
(2) **if** $d = n$ **then** /* total sort is the optimal option */
(3) totally sort list L
(4) /* successful if the knee point exists */
(5) Search the sorted list for the knee point e
(6) **return** e /* the algorithm is completed */
(7) **else** /* top-d sorting is the optimal option */
(8) QuickSortTopK(L, 1, *length*(L), d)
(9) Search the sorted list for the knee point e
(10) **if** e is found **then**
(11) **return** e
(12) **else** /* Update the parameters for the next round of optimization problem */
(13) update P according to (5) /* the posterior probability */
(14) update L by exluding the top-d item from list L
(15) **FindKnee**(L, P)

Algorithm 3: FindKnee(L, P).

then

$$c_1 n \log(p) + c_1 n \log(n) = 2c_2(1-p) n \log(1-p) + 2c_2(1-p) n \log(n). \quad (32)$$

Leave out the low order item in both sides of (32), and we get

$$c_1 n \log(p) + c_1 n \log(n) \approx c_1 n \log(n),$$
$$2c_2(1-p) n \log(1-p) + 2c_2(1-p) n \log(n) \approx 2c_2(1-p) n \log(n). \quad (33)$$

Plugging (33) into (32), the optimum of p yields

$$p = 1 - \frac{c_1}{2c_2}. \quad (34)$$

We can see that the optimum of p that takes the form of a proportion function of n exists only when $c_1 < 2c_2$. Particularly, if $c_1 = c_2 = 1$, the optimum of p is 1/2, which means that the optimum first selection number is half of the length of the list. As the search algorithm may run recursively if the knee point search misses, the optimal method is the so-called binary search or logarithmic search method.

Theorem 7. *If the probability distribution of the knee point follows uniform distribution and the optimal selection number takes the form of* (30)*, the optimal method for the knee point search algorithm is binary search or logarithmic search method.*

Proof. We first prove using an inductive method that in each recursive step of the search algorithm, $p(i)$ is equal for all $i \in N(1,n)$.

We know that for the first recursive step, $p(i)$ is equal for all $i \in N(1,n)$ as the starting point of the induction. Then for the second recursive step, we can derive according to (24) that $p(i)$ is equal for all $i \in N(1,n)$.

Suppose that for the lth recursive step, $p(i)$ is equal for all $i \in N(1,n)$. Similarly, for the $(l+1)$th recursive step, $p(i)$ is equal for all $i \in N(1,n)$ following (24).

Therefore we can form the concept inductively that in each recursive step of the knee point search algorithm, $p(i)$ is equal for all $i \in N(1,n)$. So the optimization problem in each recursive step can be written as no other than (25), whose solution of number of selection is half of the length of the list discussed above. Here the list is the one left from previous recursive step. Hence the optimal method for the search algorithm is binary search or logarithmic search method. □

(2) Inverse Proportion Distribution. $p(i)$ is in direct inverse proportion to i, $i \in N(1,n)$, and thus $p(i) = c/i$, where $c = 1/\sum_{i=1}^{n}(1/i)$.

As an approximate treatment of the summation $\sum_{i=a}^{b}(1/i)$, we consider

$$\sum_{i=a}^{b}\frac{1}{i} \approx \int_{a}^{b}\frac{1}{x}dx. \tag{35}$$

Particularly, for $a = 1$ and a large b, we have

$$\sum_{i=1}^{b}\frac{1}{i} = \ln b + E. \tag{36}$$

And for a large n, we have

$$c = 1/(\ln n + E), \tag{37}$$

where E is the Euler constant and has the approximate value as 0.5772.

Plugging $p(i)$ and then (36) and (37) into (23), we have

$$\begin{aligned} \text{Min}: \quad & C(d,n) + \left(1 - \frac{\ln d + E}{\ln n + E}\right)(S(n-d) + D(n-d)) \\ \text{s.t.} \quad & d \in N(1,n). \end{aligned} \tag{38}$$

Let

$$g(d) = C(d,n) + \left(1 - \frac{\ln d + E}{\ln n + E}\right)(S(n-d) + D(n-d)). \tag{39}$$

Theorem 8. *If the probability distribution of the knee point $p(i)$ is in direct inverse proportion to i, $i \in N(1,n)$, the optimal top-k sorting for the knee point search algorithm is full sorting of the list for a large length of the list.*

Proof. We only need to prove that $g(d)$ is a monotony increase function of d, and the equivalent condition for it is

$$\begin{aligned} \frac{dg(d)}{d} = {} & \frac{c_1(1+\log d) - c_2(n-d)\log(n-d) - c_3 n}{(\ln n + E)\,d} \\ & - \frac{c_2(\ln n - \ln d)(1+\log(n-d))}{\ln n + E} < 0, \end{aligned} \tag{40}$$

that is,

$$\begin{aligned} & c_1(1+\log d)(\ln n + E)\,d \\ & \quad < c_2(n-d)\log(n-d) \\ & \quad + c_3 n + c_2(\ln n - \ln d)(1+\log(n-d))\,d. \end{aligned} \tag{41}$$

We assume that d is an exponential function of n, such that

$$d = n^q, \quad 0 < q < 1. \tag{42}$$

Thus the left and right sides of (41) can be, respectively, written as

$$c_1(1+\log d)(\ln n + E)\,d \sim O\left((\log n)^2 n^q\right) \tag{43}$$

$$\begin{aligned} & c_2(n-d)\log(n-d) + c_3 n \\ & \quad + c_2(\ln n - \ln d)(1+\log(n-d))\,d \sim O(n). \end{aligned} \tag{44}$$

Comparing (43) and (44), we obtain (41) for a large n. That means the optimal selection number is n for the first step, or the optimal top-k sorting for the knee point search algorithm is full sorting. □

4. Source Detection of DNS DoS Flooding Attacks: An Application Example

4.1. DNS DoS Flooding Attacks. The Domain Name System is a fundamental and indispensable component of the modern Internet [13, 14]. The availability of the DNS can affect the availability of a large number of Internet applications. Ensuring the DNS data availability is an essential part of providing a robust Internet.

In the past few years, some important DNS name servers on the top level of the DNS hierarchical structure were targeted by the DoS or DDoS attackers, and some of these attacks did succeed in disabling the DNS servers and resulted in parts of the Internet experiencing severe name resolution problems [15–18]. Particularly, DNS DoS flooding attacks are the attacks launched by the attackers towards the DNS name servers with an overwhelming traffic flux in order to disrupt the DNS service for the legitimate clients. However, it is usually not easy to efficiently detect and defend the DoS flooding attacks because the attacking traffic is blended with the legitimate ones, which complicates the distinguishing efforts. Moreover, the detection mechanism should be implementable or should not add heavy computational load. Here we focus on the source-based detection method and show that the problem of source detection of DNS DoS flooding attacks can be addressed by the knee point search in the sorted curve discussed in this paper.

4.2. Detection Using the Knee Point Search. Generally, DNS name servers may receive queries coming from thousands of DNS clients (mostly DNS cache servers), whose traffic volumes are expected to remain far below those of the DoS flooding attacks. The real-time query rates for all incoming sources can be counted by the traffic monitoring system residing at the border gateway in front of the DNS name server. The goal of the DoS attack defense is to realize real-time attacking source detection and then filter out the attacking traffic from these sources accordingly. Therefore the detection problem is equivalent to knee point search in the sorted curve, where all points above the largest knee point are identified as the attacking sources. Moreover, time efficiency is also the key requirement for the problem, for timely attacking detection means timely defending action. Applying the knee point search algorithm proposed in this paper, the expected detection time is minimized.

4.3. Leaning the Knee Point Distribution. The assumption on probability distribution of the knee point is the prerequisite for the knee point search algorithm. However, in the initial rounds of detection we have hardly any a priori knowledge about the knee point. But the distribution estimation of the knee point can be learned based on the empirical data obtained in all previous rounds of detection.

First, suppose that the knee point largely follows stationary random distribution; hence its distribution exhibits almost the same probability model in all rounds of detection. We can fit a statistical model to data and provide estimates for the model's parameters. Here we apply the method of maximum likelihood for the estimation.

Let the count variable of detected knee points so far at position i in the ordered list of length n be $h_i, i = 1, 2, \ldots, n$. Let the value of h_i be x_i, $i = 1, 2, \ldots, n$. If the number of rounds of detection is R, we have

$$\sum_{i=1}^{n} x_i = R. \tag{45}$$

Let the probability vector of the knee point at different positions be $P = (p_1, p_2, \ldots, p_n)'$. The likelihood function of P can be written as

$$\text{lik}\ (P) = f_D\left(h_1 = x_1, h_2 = x_2, \ldots, h_n = x_n \mid P\right), \tag{46}$$

where $f_D(\cdot)$ is the density function. To calculate lik (P), we have

$$\begin{aligned}
\text{lik}\,(P) &= f_D\left(h_1 = x_1 \mid P\right) f_D\left(h_2 = x_2, \ldots, h_n = x_n \mid h_1 = x_1, P\right) \\
&= f_D\left(h_1 = x_1 \mid P\right) f_D\left(h_2 = x_2 \mid h_1 = x_1, P\right) f_D \\
&\quad \times \left(h_3 = x_3, \ldots, h_n = x_n \mid h_1 = x_1, h_2 = x_2, P\right) \\
&\vdots \\
&= f_D\left(h_1 = x_1 \mid P\right) f_D\left(h_2 = x_2 \mid h_1 = x_1, P\right) \cdots \\
&\quad f_D\left(h_{n-2} = x_{n-2} \mid h_1 = x_1, h_2 = x_2, \ldots, h_{n-3} = x_{n-3}, P\right) \\
&\quad \times f_D\left(h_{n-1} = x_{n-1}, h_n = x_n \mid h_1 = x_1, \right. \\
&\qquad \left. h_2 = x_2, \ldots, h_{n-2} = x_{n-2}, P\right) \\
&= f_D\left(h_1 = x_1 \mid P\right) f_D\left(h_2 = x_2 \mid h_1 = x_1, P\right) \cdots \\
&\quad f_D\left(h_{n-2} = x_{n-2} \mid h_1 = x_1\,, \right. \\
&\qquad \left. h_2 = x_2, \ldots, h_{n-3} = x_{n-3}, P\right) \\
&\quad \times f_D\left(h_{n-1} = x_{n-1} \mid h_1 = x_1, \right. \\
&\qquad \left. h_2 = x_2, \ldots, h_{n-2} = x_{n-2}, P\right) \\
&\quad \times f_D\left(h_n = x_n \mid h_1 = x_1, h_2 = x_2, \ldots, h_{n-1} = x_{n-1}, P\right),
\end{aligned} \tag{47}$$

where the last item in (47) is actually not an independent one given that all h_i other than h_n are known due to the constraint in (45), such that

$$f_D\left(h_n = x_n \mid h_1 = x_1, h_2 = x_2, \ldots, h_{n-1} = x_{n-1}, P\right) = 1. \tag{48}$$

Plugging (48) into (47), we get

$$\begin{aligned}
\text{lik}\ (P) &= \binom{n}{x_1} p_1^{x_1} (1 - p_1)^{n - x_1} \binom{n - x_1}{x_2} \left(\frac{p_2}{1 - p_1}\right)^{x_2} \\
&\quad \times \left(1 - \frac{p_2}{1 - p_1}\right)^{n - x_1 - x_2} \cdots \binom{n - \sum_{i=1}^{n-2} x_i}{x_{n-1}} \\
&\quad \times \left(\frac{p_{n-1}}{1 - \sum_{i=1}^{n-2} p_i}\right)^{x_{n-1}} \left(1 - \frac{p_{n-1}}{1 - \sum_{i=1}^{n-2} p_i}\right)^{n - \sum_{i=1}^{n-1} x_i}.
\end{aligned} \tag{49}$$

Thus let

$$\frac{\partial \ln\left(\text{lik}\,(P)\right)}{\partial p_i} = 0, \quad i = 1, 2, \ldots, n - 1. \tag{50}$$

We obtain the maximum likelihood estimation of p_i, $i = 1, 2, \ldots, n - 1$:

$$\begin{aligned}
\widehat{p}_i &= \frac{x_i}{R}, \quad i = 1, 2, \ldots, n - 1, \\
\widehat{p}_n &= 1 - \sum_{i=1}^{n-1} \widehat{p}_i = \frac{x_n}{R}.
\end{aligned} \tag{51}$$

At the beginning of each round of detection, if the previous round finds the knee point at position i^*, R and x_i, $i = 1, 2, \ldots, n$, are updated as follows:

$$\begin{aligned}
R &\longleftarrow R + 1, \\
x^{i*} &\longleftarrow x^{i*} + 1, \\
x^i &\longleftarrow x^i, \quad i \in N\,(1, n)\,,\ i \neq i^*.
\end{aligned} \tag{52}$$

The knee point distribution may evolve over time; thus the position of the knee point detected in recent rounds

provides more reliable information for the estimation than earlier rounds. Taking the chronological order into consideration, we assign more weight to recent rounds than earlier rounds. This can be done by decreasing the detection results in previous rounds progressively. The deceasing is performed in updating R and x_i, $i = 1, 2, \ldots, n$, and sums up the current detection and the previous ones at a discount β, $0 < \beta < 1$. Formally, the updating of R and x_i, $i = 1, 2, \ldots, n$, can be modified as follows:

$$\begin{aligned} R &\longleftarrow \beta^* R + 1, \\ x^{i*} &\longleftarrow \beta^* x^{i*} + 1, \\ x^i &\longleftarrow \beta * x^i, \quad i \in N(1, n),\ i \neq i^*. \end{aligned} \tag{53}$$

5. Conclusion

Knee point search in the sorted curve is often used in the practice of anomaly detection and many other applications. Due to the inefficiency of total sorting, top-k sorting should be adopted for the knee point search. In this paper, a knee point search algorithm using cascading top-k sorting is proposed. The expected time complexity is minimized via optimizing the selection number k in each step.

Acknowledgments

This work was supported in part by the National Key Technology R&D Program of China under Grant no. 2012BAH16B00 and the National Science Foundation of China under Grant no. 61003239.

References

[1] J. M. Kang, S. Shekhar, C. Wennen, and P. Novak, "Discovering flow anomalies: a SWEET approach," in *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM '08)*, pp. 851–856, December 2008.

[2] H. Hajji, "Statistical analysis of network traffic for adaptive faults detection," *IEEE Transactions on Neural Networks*, vol. 16, no. 5, pp. 1053–1063, 2005.

[3] D. Gao, M. K. Reiter, and D. Song, "Beyond output voting: detecting compromised replicas using HMM-based behavioral distance," *IEEE Transactions on Dependable and Secure Computing*, vol. 6, no. 2, pp. 96–110, 2009.

[4] X. Li, F. Bian, M. Crovella et al., "Detection and identification of network anomalies using sketch subspaces," in *Proceedings of the 6th ACM SIGCOMM on Internet Measurement Conference (IMC '06)*, pp. 147–152, October 2006.

[5] S. Salvador and P. Chan, "Learning states and rules for detecting anomalies in time series," *Applied Intelligence*, vol. 23, no. 3, pp. 241–255, 2005.

[6] Q. Zhao, V. Hautamaki, and P. Fränti, "Knee point detection in BIC for detecting the number of clusters," in *Advanced Concepts for Intelligent Vision Systems*, vol. 5259 of *Lecture Notes in Computer Science*, pp. 664–673, Springer, Berlin, Germany, 2008.

[7] C. A. R. Hoare, "Quicksort," *Computer Journal*, vol. 5, no. 1, pp. 10–15, 1962.

[8] D. Knuth, *Sorting and Searching*, vol. 3 of *The Art of Computer Programming*, Addison-Wesley, New York, NY, USA, 3rd edition, 1997.

[9] H. M. Okasha and U. Rösler, "Asymptotic distributions for random median quicksort," *Journal of Discrete Algorithms*, vol. 5, no. 3, pp. 592–608, 2007.

[10] J. Pagter and T. Rauhe, "Optimal time-space trade-offs for sorting," in *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, pp. 264–268, November 1998.

[11] D. Cantone and G. Cincotti, "QuickHeapsort, an efficient mix of classical sorting algorithms," *Theoretical Computer Science*, vol. 285, no. 1, pp. 25–42, 2002.

[12] J. Cardinal, S. Fiorini, G. Joret, R. M. Jungers, and J. I. Munro, "Sorting under partial information (without the ellipsoid algorithm)," in *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC '10)*, pp. 359–368, June 2010.

[13] P. Mockapetris, "Domain names—concepts and facilities," Internet Request for Comments (RFC, 1034), November 1987.

[14] P. Albitz and C. Liu, *DNS and BIND*, O'Reilly and Associates, 1998.

[15] Name server DoS Attack October, 2002, http://www.caida.org/projects/dns-analysis/.

[16] UltraDNS DOS Attack, 2002, http://www.theregister.co.uk/2002/12/14/ddos_attack_really_really_tested/.

[17] "DoS Attack against Akamai," 2004, http://news.cnet.com/2100-1038_3-5236403.html.

[18] "ICANN Factsheet for the February 6, 2007 Root Server Attack," 2007, http://www.icann.org/announcements/factsheet-dns-attack-08mar07_v1.1.pdf.