# Dynamic Interpolation Search Revisited *

Alexis Kaporis[1,2], Christos Makris[1], Spyros Sioutas[1], Athanasios Tsakalidis[1,2], Kostas Tsichlas[1], and Christos Zaroliagis[1,2]

[1] Dept of Computer Eng and Informatics, University of Patras, 26500 Patras, Greece
[2] Computer Technology Institute, N. Kazantzaki Str, Patras University Campus, 26500 Patras, Greece.
Email: {kaporis,makri,sioutas,tsak,tsihlas,zaro}@ceid.upatras.gr

**Abstract.** A new dynamic Interpolation Search (IS) data structure is presented that achieves $O(\log\log n)$ search time with high probability on unknown continuous or even discrete input distributions with measurable probability of key collisions, including power law and Binomial distributions. No such previous result holds for IS when the probability of key collisions is measurable. Moreover, our data structure exhibits $O(1)$ expected search time with high probability for a wide class of input distributions that contains all those for which $o(\log\log n)$ expected search time was previously known.

## 1 Introduction

The dynamic dictionary search problem is one of the fundamental problems in computer science. In this problem we have to maintain a set of elements subject to insertions and deletions such that given a query element $y$ we can retrieve the largest element in the set smaller or equal to $y$. Well known search methods use an arbitrary rule to *select* a splitting element and *split* the stored set into two subfiles; in binary search, each recursive split selects as splitting element, in a "blind" manner, the middle (or a close to the middle) element of the current file. Using this technique, known balanced search trees (e.g., $(a,b)$-trees [11]) support search and update operations in $O(\log n)$ time when storing $n$ elements. In the Pointer Machine (PM) model of computation, the search time cannot be further reduced, since the lower bound of $\Omega(n\log n)$ for sorting $n$ elements would be violated. In the RAM model of computation, which we consider in this work, a lower bound of $\Omega(\sqrt{\frac{\log n}{\log\log n}})$ was proved by Beame and Fich [4]; a data structure achieving this time bound has been presented by Andersson and Thorup [2].

The aforementioned lower bounds can be surpassed if we take into account the input distribution of the keys and consider expected complexities; in this case, the extra knowledge about the probabilistic nature of the keys stored in the file

* This work was partially supported by the FET Unit of EC (IST priority – 6th FP), under contracts no. IST-2002-001907 (integrated project DELIS) and no. FP6-021235-2 (project ARRIVAL), and by the Action PYTHAGORAS with matching funds from the European Social Fund and the Greek Ministry of Education.

may lead to better selections of splitting elements. The main representative of these techniques is the method of *Interpolation Search* (IS) introduced by Peterson [21], where the splitting element was selected close to the expected location of the target key. Yao and Yao [28] proved a $\Theta(\log\log n)$ average search time for stored elements that are uniformly distributed. In [9, 10, 18–20] several aspects of IS are described and analyzed. Willard [26] proved the same search time for the extended class of *regular* input distributions. The IS method was recently generalized [5] to non-random input data that possess enough "pseudo-randomness" for effective IS to be applied. The study of dynamic insertions of elements with respect to the uniform distribution and random deletions was initiated in [8, 12]. In [8] an implicit data structure was presented supporting insertions and deletions in $O(n^{\epsilon}), \epsilon > 0$, time and IS with expected time $O(\log\log n)$. The structure of [12] has expected insertion time $O(\log n)$, amortized insertion time $O(\log^2 n)$ and it is claimed, without rigorous proof, that it supports IS. Mehlhorn and Tsakalidis [16] demonstrated a novel dynamic version of the IS method, the *Interpolation Search Tree (IST)*, with $O(\log\log n)$ expected search and update time for a larger class than the regular distributions. In particular, they considered $\mu$-random insertions and random deletions[3] by introducing the notion of a $(f_1, f_2)$*-smooth* probability density $\mu$, in order to control the distribution of the elements in each subinterval dictated by an ID index. Informally, a distribution defined over an interval $I$ is smooth if the probability density over any subinterval of $I$ does not exceed a specific bound, however small this subinterval is (i.e., the distribution does not contain sharp peaks). The class of smooth distributions is a superset of uniform, bounded, and several non-uniform distributions (including the class of regular distributions). The results in [16] hold for $(n^{\alpha}, \sqrt{n})$-smooth densities, where $1/2 \leq \alpha < 1$. Andersson and Mattson [1], generalized and refined the notion of smooth distributions, presenting a variant of the IST called *Augmented Sampled Forest* extending the class of input distributions for which $\Theta(\log\log n)$ search time is expected. In particular, the time complexities of their structure holds for the larger class of $(\frac{n}{(\log\log n)^{1+\epsilon}}, n^{\delta})$-smooth densities, where $\delta \in (0,1), \epsilon > 0$. Moreover, their structure exhibited $o(\log\log n)$ expected search time for some classes of input distributions. Finally in [13], a finger search version of these structures was presented.

The analysis of all the aforementioned IS structures was heavily based on the assumption that the conditional distribution on the subinterval dictated by an arbitrary interpolation step remains unaffected. In particular, in [1, 13, 16] IS is performed on each node of a tree structure under the assumption that all elements in the subtree dictated by the previous interpolation step remain $\mu$-random.

Our first contribution in this work (Section 2) is to show that the above assumption is valid only when the produced elements are *distinct* (as indeed assumed in [1, 9, 10, 13, 16, 19–21, 26, 28]), i.e., they are produced under some con-

[3] An insertion is $\mu$-random if the key to be inserted is drawn randomly with density function $\mu$; a deletion is random if every key present in the data structure is equally likely to be deleted.

tinuous distribution where the probability of collision is zero; otherwise, it *fails*. This means that the probabilistic analyses of previous dynamic interpolation search data structures are inapplicable to sequences of non-distinct elements, produced by discrete probability distributions with measurable (non-zero) probability of key collisions.

This lack of generalization does not have only theoretical, but also serious practical implications. There exist applications where we need to store duplicates, and thus the theoretically used density distribution modelling the input process should *not* produce distinct elements. A classical example is the creation of secondary indices in databases [15]. In a secondary index, duplicate values correspond to different records and they should be stored as distinct entities. There are also specific applications where interpolation search comes into play. For instance, the case of searching tables with alphabetic keys (e.g., names, dictionary entries) [18]. The keys in such tables follow a non-uniform, (unknown) discrete probability distribution and collisions *do* occur. Other useful applications of interpolation search in non-uniform data are discussed in [3, 7, 18, 20, 22]. In these papers it has been empirically observed that interpolation search has a very poor performance in such data. To alleviate this problem a series of heuristics have been introduced in [3, 7, 18, 20, 22], but no rigorous performance analyses have been provided. In [18, 19], it was suggested that such an analysis would be possible if one considers the idea in [10] that translates any continuous input distribution to a uniform one.

In Section 2, we also show that this idea of taking advantage of the cumulative distribution [10, 18, 19] does *not* apply to discrete distributions with measurable probability of key collisions (a fact that was indeed experimentally verified in [18]). The above pluralism of efforts demonstrates the necessity to handle non-uniform data generated by discrete distributions with measurable probability of key collisions.

One could be tempted to argue that the inapplicability of the previous analyses could be faced by simply storing duplicate elements once; moreover, in these structures the main rebalancing tool is local/global rebuilding, which can be easily modified to produce input sequences with distinct elements. Both arguments are wrong, however, since the new sequences of distinct elements are artificial sequences, different from the initial. Consequently, important statistical properties of the elements are destroyed and the probabilistic analyses fail.

Our second contribution in this paper is a new dynamic interpolation search data structure (Section 3) that overcomes the above problems, and in which the elements stored in each subtree preserve the input distribution, conditioning only on the interval that corresponds to the current subtree. The new structure is quite simple, it exhibits similar expected $O(\log \log n)$ search time as the previous dynamic interpolation structures [1, 9, 10, 16, 19–21, 26, 28]), its probabilistic analysis is *always* valid irrespectively of the distinctness or not of the elements in the input sequence (i.e., *regardless* of whether they are produced by a continuous or a discrete distribution), it applies to the same classes of distributions as those in [1, 16] and it holds with high probability, while those in [1, 9, 10, 16, 19–21, 26, 28]) did not grant such guarantee. Finally, as a by-product of our construction, we get a dynamic search data structure with $O(1)$ expected search time for a wide class of input distributions (Section 3). This result significantly extends the class of input distributions in [1] under which $O(1)$ expected search time was possible. In addition, this search time also holds with high probability, while those in [1] did not grant such property.

Although the class of smooth distributions includes, for appropriate choices of $f_1$ and $f_2$, any other probability distribution, the effective range of $f_1$, $f_2$ for which $O(\log\log n)$ IS time is achieved excludes distributions of major practical importance; for instance, power law [17], Binomial, etc. We are able to show (Section 3) that a slight modification of our data structure achieves $O(\log\log n)$ time with high probability for power law and Binomial distributions. No previous IS structure achieves such a time bound for these distributions (recall the deterioration of IS that was experimentally observed in [3, 7, 18, 20, 22]).

Our data structure is *robust* (as those in [1, 13, 16, 26]), i.e., it remains efficient *without* apriori knowledge of the particular continuous or discrete distribution. Due to space limitations, several details and proofs are omitted and can be found in the full version [14].

## 2 Probabilistic Analysis of the IS-tree Revisited

Consider an unknown *continuous* probability distribution over the interval $[a,\ b]$ with density function $\mu(x) = \mu[a,b](x)$. Given two functions $f_1$ and $f_2$, then $\mu(x) = \mu[a,b](x)$ is $(f_1, f_2)$-*smooth* [1, 16] if there exists a constant $\beta$, such that for all $c_1, c_2, c_3$, $a \le c_1 < c_2 < c_3 \le b$, and all integers $n$, it holds that

$$\Pr[X \in [c_2 - \frac{c_3 - c_1}{f_1(n)}, c_2]\ |c_1 \le X \le c_3] = \int_{c_2 - \frac{c_3 - c_1}{f_1(n)}}^{c_2} \mu[c_1, c_3](x)dx \le \frac{\beta f_2(n)}{n} \quad (1)$$

where $\mu[c_1, c_3](x) = 0$ for $x < c_1$ or $x > c_3$, and $\mu[c_1, c_3](x) = \mu(x)/p$ for $c_1 \le x \le c_3$ where $p = \int_{c_1}^{c_3} \mu(x)dx$. Similarly, for an unknown *discrete* probability distribution of elements $x_1, \ldots, x_N$ spread over $[a,\ b]$, with probability function $\mu(x_i) = \mu[a, b](x_i)$ we have

$$\Pr[c_2 - \frac{c_3 - c_1}{f_1(n)} \le X \le c_2 | c_1 \le X \le c_3] = \sum_{c_2 - \frac{c_3 - c_1}{f_1(n)}}^{c_2} \mu[c_1,\ c_3](x_i) \le \frac{\beta f_2(n)}{n} \quad (2)$$

where $\mu[c_1, c_3](x_i) = 0$ for $x_i < c_1$ or $x_i > c_3$, and $\mu[c_1,\ c_3](x_i) = \mu(x_i)/p$ for $x_i \in [c_1,\ c_3]$ where $p = \sum_{x_i \in [c_1,\ c_3]} \mu(x_i)$. Intuitively, function $f_1$ partitions an arbitrary subinterval $[c_1, c_3] \subseteq [a, b]$ into $f_1$ equal parts, each of length $\frac{c_3 - c_1}{f_1} = O(\frac{1}{f_1})$; that is, $f_1$ measures how *fine* is the partitioning of an arbitrary subinterval. Function $f_2$ guarantees that no part, of the $f_1$ possible, gets more probability mass than $\frac{\beta \cdot f_2}{n}$; that is, $f_2$ measures the *sparseness* of any subinterval $[c_2 - \frac{c_3 - c_1}{f_1}, c_2] \subseteq [c_1, c_3]$. The class of $(f_1, f_2)$-smooth distributions (for appropriate choices of $f_1$ and $f_2$) is a superset of both regular and uniform classes of distributions, as well as of several non-uniform classes [1, 16]. Actually, *any* probability distribution is $(f_1, \Theta(n))$-smooth, for a suitable choice of $\beta$.

Consider the random file $S = \{X_1, \ldots, X_n\}$, where each key $X_i \in [a, b] \subset I\!R$, obeys an unknown (discrete or continuous) distribution $\mu$, $i = 1, \ldots, n$. Let $P = \{X_{(1)}, \ldots, X_{(n)}\}$ be an increasing ordering of file $S$. The goal is to find the largest key $X_{(j)} \in P$ that precedes a *target* element $y$. We describe how the *Augmented Sampled Forest (ASF)* [1], which is a generalization of the *Interpolation Search Tree (IST)* [16], can be used to search for this target element $y$.

Assume that the (discrete or continuous) distribution $\mu$ is $(I(n), n/R(n))$-*smooth*, where $I(n)$, $R(n)$ are two nondecreasing functions. The ASF is a two level data structure; the top level is an *ideal* static IST [16] while the bottom level is a sequence of buckets. The structure is maintained by using the global rebuilding technique and its expected search time is dominated by the expected search time at the top level. At the top level, the root node has $R(n)$ children, and similarly each child node has $R(\frac{n}{R(n)})$ sub-children. The root node corresponds to the ordered file $P$ of size $n$. Each child corresponds to a part of file $P$ of size $\frac{n}{R(n)}$. That is, these $R(n)$ children partition the ordered file $P$ into $R(n)$ equal subfiles $P_1, \ldots, P_{R(n)}$, of the form $\{X_{(1)}, \ldots, X_{(\frac{n}{R(n)})}\}, \ldots, \{X_{((R(n)-1)\frac{n}{R(n)}+1)}, \ldots, X_{(n)}\}$. Each node of this tree contains a pair of arrays, namely ID and REP, that help to locate the appropriate child eligible to contain the target element $y$. In the root node the set of indices of the ID array is $[1, \ldots, I(n)]$ and the set of indices of the REP array is $[1, \ldots, R(n)]$. The role of the ID array of the root node is to partition the interval $[a, b]$ into $I(n)$ equal parts, each of length $\frac{b-a}{I(n)}$. When searching for an element $y$, the first *interpolation* step determines within $O(1)$ time the number $j$

$$j = \left\lfloor \frac{y-a}{b-a} I(n) \right\rfloor + 1 \tag{3}$$

which denotes the $j$-th interval $I_j$ of length $\frac{b-a}{I(n)}$ that contains the target $y$:

$$I_j = \left[ a + (j-1)\frac{b-a}{I(n)},\ a + j\frac{b-a}{I(n)} \right] \tag{4}$$

The role of the array REP$[1, \ldots, R(n)]$ of the root node is to partition the ordered file $P$ into $R(n)$ equal subfiles, each of size $\frac{n}{R(n)}$. Index REP$[i], i = 1, \ldots, R(n)$, points to the $i$-th subfile $P_i$, where $P_i = \{X \in P \mid X_{((i-1)\frac{n}{R(n)})} < X \leq X_{(i\frac{n}{R(n)})}\}$. Alternatively, REP$[i]$ can be seen as the *representative* of the element $X_{(i\frac{n}{R(n)})}$ of $P_i$. The first interpolation step, provided by Eq. (3), determines within $O(1)$ time the subinterval $I_j$ described in Eq. (4), where the target element $y$ belongs. If in this subinterval correspond $O(1)$ REP indices, then within $O(1)$ time we can determine the unique REP index that corresponds to the subfile that element $y$ may belong. Hence, the search efficiency highly depends on the distribution of the REP indices over each ID subinterval of $[a, b]$. In other words, each ID index that corresponds to a *dense* subinterval of $[a,\ b]$ causes a great slow-down of the search speed. Most importantly, suppose that the second interpolation step now yields REP$[s-1] < y \leq$REP$[s]$. Then, $y$ must be searched for into the subfile $P_s = \{X_{((s-1)\frac{n}{R(n)})}, \ldots, X_{(s\frac{n}{R(n)})}\}$. A crucial observation is that its endpoints $X_{((s-1)\frac{n}{R(n)})}, X_{(s\frac{n}{R(n)})}$ may in general be neither $\mu$-random nor smooth.

The analyses in [1, 13, 16] assume that the elements into an arbitrary subfile dictated by an interpolation step *remain $\mu$-randomly distributed* conditioned on the subinterval that all these elements belong, i.e., for a random element $X = \lambda$ in subfile $P_v$ with endpoints $a' = X_{(v-1)\frac{n}{R(n)}}$ and $b' = X_{v\frac{n}{R(n)}}$, its probability density is given by Expression (5). Also, the analyses in [10, 18, 19] ingeniously apply the cumulative distribution function $F$ on the ordered keys in $P = \{X_{(1)}, \ldots, X_{(n)}\}$, yielding $P_F = \{F(X_{(1)}), \ldots, F(X_{(n)})\}$. Now, each $F(X_{(i)}) \in P_F$ is *uniformly* distributed over $[0,1]$, since $\Pr[F(X_{(i)}) \leq t] = \Pr[X_{(i)} \leq F^{-1}(t)] = F(F^{-1}(t)) = t$ (see [6, pp. 36-37]). Thus, file $P_F$ is very suitable for applying IS on it; i.e., to search for target key $y$, split $P_F$ on key $F(X_{(j_y)}) \approx \frac{y - F(X_{(1)})}{F(X_{(n)}) - F(X_{(1)})}$, and recursively apply IS to $P_F^- = \{F(X_{(1)}), \ldots, F(X_{(j_y)})\}$, if $y \leq F(X_{(j_y)})n$, otherwise to $P_F^+ = P_F \setminus P_F^-$. However, this approach *also* tacitly assumes that the conditional distribution of the keys in subfiles $P_F^-, P_F^+$ remains unaffected and obeys Expression (5) with $a', b'$ the corresponding endpoints of the appropriate subfile $P_F^-$ or $P_F^+$.

In the following, we prove the validity of these assumptions under continuous or discrete distributions with zero probability of element collisions, and we will depict the subtle case of discrete distributions with measurable probability of key-collisions where all the above assumptions *fail*.

**Continuous or discrete distributions with zero probability of element collisions.** Consider the simple case of three stored elements (random variables) $X_1, X_2, X_3 \in [a,b]$ drawn according to some $\mu$-random smooth distribution (the general case of $n$ variables can be easily deduced from this case by a simple induction argument). These elements are identically and independently distributed and it is assumed that they take *distinct* values. Since the collision probability for continuous distributions is 0, we concentrate our discussion to distinct elements. The *conditional*, on the arbitrary interval with *fixed* endpoints $(a', b'] \subseteq [a,b]$, probability density equals

$$\Pr[X = \lambda \mid a' < X \leq b'] = \frac{\Pr[X = \lambda]}{\Pr[X \leq b'] - \Pr[X \leq a']}. \tag{5}$$

According to definitions (1) and (2), Exp. (5) plays a crucial role in tuning the probability mass in subinterval $(a', b']$ using parameters $f_1, f_2$. For each $i = 1, 2$ the corresponding REP$[i]$ is a *new* random variable defined as REP$[1] \equiv X_{(1)} = \min\{X_1, X_2, X_3\}$, REP$[2] \equiv X_{(3)} = \max\{X_1, X_2, X_3\}$. We want to show that the random element $X$ that belongs into the subinterval [REP[1], REP[2]] is $\mu$-randomly distributed. We have

$$\begin{aligned}\Pr[X = \lambda \mid \text{REP}[1] = a' < X \leq \text{REP}[2] = b'] &= \Pr[X = \lambda \mid X_{(1)} = a' \cap X_{(3)} = b'] \\ &= \frac{\Pr[X = \lambda \cap X_{(1)} = a' \cap X_{(3)} = b']}{\Pr[X_{(1)} = a' \cap X_{(3)} = b']},\end{aligned} \tag{6}$$

where $a' < \lambda < b'$. The event $\{X_{(1)} = a' \cap X_{(3)} = b'\}$ occurs if at least one of the following mutually disjoint events occur:

$$\begin{gathered}
\{X_1 = a',\ X_2 = b',\ a' < X_3 < b'\},\ \{X_2 = a',\ X_1 = b',\ a' < X_3 < b'\},\\
\{X_1 = a', X_3 = b',\ a' < X_2 < b'\},\ \{X_3 = a',\ X_1 = b',\ a' < X_2 < b'\},\\
\{X_2 = a',\ X_3 = b', a' < X_1 < b'\},\ \{X_3 = a',\ X_2 = b',\ a' < X_1 < b'\}.
\end{gathered} \tag{7}$$

$$\text{Hence,}\quad \Pr[X_{(1)} = a' \cap X_{(3)} = b'] = 6\Pr[X = a']\Pr[X = b']\Pr[a' < X < b'] \tag{8}$$

Similarly, the event $\{X = \lambda\ \cap\ X_{(1)} = a'\ \cap\ X_{(3)} = b'\}$, with $a' < \lambda < b'$, occurs if one of the following mutually disjoint events occur:

$$\begin{gathered}
\{X_2 = \lambda,\ X_3 = a',\ X_1 = b'\},\ \{X_1 = \lambda,\ X_2 = a',\ X_3 = b'\},\\
\{X_3 = \lambda,\ X_1 = a', X_2 = b'\},\ \{X_1 = \lambda,\ X_3 = a',\ X_2 = b'\},\\
\{X_3 = \lambda,\ X_2 = a',\ X_1 = b'\},\ \{X_2 = \lambda, X_1 = a',\ X_3 = b'\}.
\end{gathered} \tag{9}$$

Combining (8) and (9), the conditional probability (6) becomes

$$\begin{aligned}
\Pr[X = \lambda | X_{(1)} = a' \cap X_{(3)} = b'] &= \frac{6\Pr[X = \lambda]\Pr[X = a']\Pr[X = b']}{6\Pr[X = a']\Pr[X = b']\Pr[a' < X < b']}\\
&= \frac{\Pr[X = \lambda]}{\Pr[a' < X < b']}
\end{aligned} \tag{10}$$

where $a \leq a' < \lambda < b' \leq b$. This probability equals Exp. (5) and thus is $\mu$-random and consequently smooth (due to definitions (1) and (2)). Hence, we have shown that in the case where the input elements have non measurable probability of collisions, all previous analyses carry over correctly.

**Discrete distributions with measurable probability of element collisions.** In this case, the event $\{X_{(1)} = a' \cap X_{(3)} = b'\}$ occurs if, besides the events listed in (7), at least one of the following mutually disjoint events occur:

$$\begin{gathered}
\{X_{1,2} = a',\ X_3 = b'\}, \{X_{1,2} = b',\ X_3 = a'\}, \{X_{1,3} = a',\ X_2 = b'\},\\
\{X_{1,3} = b', X_2 = a'\}, \{X_{2,3} = a',\ X_1 = b'\}, \{X_{2,3} = b',\ X_1 = a'\}
\end{gathered} \tag{11}$$

$$\begin{aligned}
\text{Hence,}\quad \Pr[X_{(1)} = a' \cap X_{(3)} = b'] &= 3\Pr[X = a']^2\Pr[X = b'] +\\
3\Pr[X = a']\Pr[X = b']^2 &+ 6\Pr[X = a']\Pr[X = b']\Pr[a' < X < b']
\end{aligned} \tag{12}$$

If $a \leq a' < \lambda < b' \leq b$, by combining (11) and (12), now Expression (6) becomes

$$\Pr[X = \lambda | X_{(1)} = a' \cap X_{(3)} = b'] = \frac{\Pr[X = \lambda]}{\frac{\Pr[X = a'] + \Pr[X = b']}{2} + \Pr[a' < X < b']} \tag{13}$$

and if $\lambda = a'$ or $\lambda = b'$, Expression (6) becomes the half of Expression (13). Clearly, (13) is different from (5) and in general may be *neither* $\mu$-random *nor* smooth (see the full version of the paper [14] for more details). We conclude that, when the probability of collisions is measurable, the net effect of choosing, as endpoints of subintervals, not deterministically obtained values is to *destroy* the smoothness of the distribution of the elements that belong in it.

## 3 The new IS data structure

Consider a dynamic file $S$ containing $O(n)$ elements drawn from the interval $[a, b]$, according to a continuous or discrete distribution $\mu$, which is $(f_1, f_2) = (n^{\alpha}, n^{\delta})$-smooth with *arbitrary* $\alpha, \delta \in (0, 1)$. Our structure consists of LAYERS of bins. The 1st LAYER partitions interval $[a, b]$ into $f_1(n)$ equal-length bins. We define[4] as $\mathrm{BIN}(j_1)$, the $j_1$-th bin in the 1st LAYER of bins, which corresponds to the subinterval $[a + (j_1 - 1)\frac{b-a}{f_1(n)},\ a + j_1\frac{b-a}{f_1(n)}] = [a_{j_1},\ b_{j_1}] \subset [a,\ b], j_1 = 1, \dots, f_1(n)$. Any key $X \in S$ is stored in $\mathrm{BIN}(j_1)$, iff $X$ is spread according to $\mu$ into the subinterval $[a_{j_1},\ b_{j_1}], j_1 = 1, \dots, f_1(n)$. This subfile $S_{j_1} \subseteq S$ consists of $n_{j_1}$ elements and is stored in $\mathrm{BIN}(j_1)$, where $n_1 + \ldots + n_{f_1(n)} = |S| = O(n)$, and $j_1 = 1, \dots, f_1(n) = n^{\alpha}$.

The 2nd LAYER of bins is constructed by recursively partitioning each $\mathrm{BIN}(j_1)$ of the 1st LAYER into $f_1(n_{j_1})$ equal-length bins, $j_1 = 1, \dots, f_1(n)$, i.e., $\mathrm{BIN}(j_1)$ containing $n_{j_1}$ elements is partitioned into equal-length bins $\mathrm{BIN}(j_1, j_2)$, with corresponding indices $j_1 = 1, \dots, f_1(n) = n^{\alpha}$ and $j_2 = 1, \dots, f_1(n_{j_1}) = (n_{j_1})^{\alpha}$. Now $\mathrm{BIN}(j_1, j_2)$ corresponds to the subinterval $[a_{j_1} + (j_2 - 1)\frac{b_{j_1} - a_{j_1}}{f_1(n_{j_1})}, a_{j_1} + j_2\frac{b_{j_1} - a_{j_1}}{f_1(n_{j_1})}] = [a_{j_1,j_2},\ b_{j_1,j_2}] \subset [a_{j_1},\ b_{j_1}] \subset [a, b]$. An arbitrary element $X \in S$ is stored in $\mathrm{BIN}(j_1, j_2)$, iff $X$ is spread according to $\mu$ into the subinterval $[a_{j_1,j_2},\ b_{j_1,j_2}],\ j_2 = 1, \dots, f_1(n_{j_1})$ and $j_1 = 1, \dots, f_1(n)$. The subfile $S_{j_1,j_2} \subseteq S_{j_1}$ consists of $n_{j_1,j_2}$ elements stored in $\mathrm{BIN}(j_1, j_2)$, such that $n_{j_1,1} + \ldots + n_{j_1, f_1(n_{j_1})} = |S_{j_1}| = n_{j_1}$.

We proceed recursively for the subsequent LAYERS of bins; however, no bin with less than poly $\log n$ keys becomes further partitioned ($n$ is the initial number of keys in the structure), i.e., it becomes a leaf of the structure. Finally, the elements associated with each leaf bin are stored as a $q^*$-heap. The *$q^*$-heap* [27] is a search tree data structure having the following useful property: let $M$ be the current number of elements in the $q^*$-heap and let $N$ be an upper bound on the maximum number of elements ever stored in the $q^*$-heap. Then, insertion, deletion and search operations are carried out in $O(1 + \log M / \log\log N)$ worst-case time after an $O(N)$ preprocessing overhead. Choosing $M = \mathrm{polylog}(N)$, all operations can be performed in $O(1)$ time. Hence, by setting $N$ to be $n$, the use of *$q^*$-heap* at the leaves of the structure permits the manipulation of search and update operations in the leaf bins in worst-case $O(1)$ time.

In the above data structure, we can search for a target element $y$ as follows. Given that a bin containing $y$ at the current LAYER has been located, we perform interpolation search on its offspring of bins to locate the particular bin of the next LAYER that $y$ may belong. Since target $y$ may belong in at most one bin of each LAYER, as the LAYERS evolve, this process highly prunes the size of the search space (the occupancy number of the currently scanned bin).

The careful reader should have noticed that the endpoints selected as representatives in each subtree are independent of the particular characteristics of the input distribution $\mu$, thus confronting the weakness of the constructions in

[4] From now on, the subscript $i$ of $j_i$ will denote the $i$-th LAYER of bins.

all previous approaches. This crucial randomness invariance property of the new data structure is given by Lemma 1 (whose proof is in [14]).

**Lemma 1.** *Consider an arbitrary bin $BIN(j_1, \ldots, j_i)$ with corresponding subinterval $[a_{j_1,\ldots,j_i},\ b_{j_1,\ldots,j_i}]$ of the ith* LAYER *of bins. Then, the $n_{j_1,\ldots,j_i}$ elements in $BIN(j_1, \ldots, j_i)$ are $\mu$-randomly distributed in the subinterval $[a_{j_1,\ldots,j_i},\ b_{j_1,\ldots,j_i}]$.*

Theorem 1 below shows that w.h.p. each IS step prunes drastically the size of the dictated subfile (its proof is in [14]), i.e., a child bin has size at most $f_2$(elements of father bin).

**Theorem 1.** *Consider the bin $BIN(j_1, \ldots, j_i)$ of the i-th* LAYER *of bins and let $n_{j_1,\ldots,j_i}$ be its number of balls at the end of the t-th insertion/deletion operation. These balls are $\mu$-randomly distributed in its subinterval $[a_{j_1,\ldots,j_i},\ b_{j_1,\ldots,j_i}]$. Then,*

$$\Pr[\exists\ BIN(j_1, \ldots, j_i, j_{i+1}) :\ n_{j_1,\ldots,j_i,j_{i+1}} = \omega(f_2(n_{j_1,\ldots,j_i}))] \to 0, \ \text{as } n \to \infty,$$

*where $j_{i+1} = 1, \ldots, f_1(n_{j_1,\ldots,j_i})$.*

This in turn yields the search time bound in Lemma 2 below (its proof in [14]).

**Lemma 2.** *For every target element y, the path from its leaf bin to the root of the tree will have length not exceeding* $\log\log n$ *with high probability.*

Moreover, for every node $v$ of the tree, the subtree of any of its children will have at most half the size of the subtree of $v$, with high probability. We call a tree with these properties *ideal*; our high probability bound implies that for a given set of $\mu$ random elements with cardinality $n$, such a tree can be found and be built in $O(n)$ expected time. Moreover, by using the arguments in [16, Lemma 2, p. 626], we can straightforwardly show that the space complexity of the described data structure is linear.

Consequently, by embedding the *ideal* version of our new data structure as the top level in the *Augmented Sampled Forest (ASF)* of [1] and by maintaining the leaf bins as a $q^*$-heap [27], while keeping in parallel a worst-case data structure [2] (in a manner e.g., similar to [13]), we get the following theorem.

**Theorem 2.** *Consider a file with n (not necessarily distinct) elements that was produced by a sequence of $\mu$-random insertions and random deletions, where $\mu$ is a $(n^\alpha, n^\delta)$-smooth density, for any arbitrary $0 < \alpha, \delta < 1$. Then, there exists a dynamic interpolation search tree with $O(\log\log n)$ expected search time with high probability; the space usage of the data structure is $\Theta(n)$, the worst-case update time (position given) is $O(1)$, and the worst-case search time is $O(\sqrt{\log n / \log\log n})$.*

**Remark.** It is easy to see that every part of our analysis remains valid if we replace the function $f_1(n) = n^\alpha$ with the function $f_1(n) = \frac{n}{(\log\log n)^{1+\epsilon}}$, where $\epsilon > 0$. Hence, our structure can handle within the same time and space complexities, as those mentioned in Theorem 2, the larger class of $(\frac{n}{(\log\log n)^{1+\epsilon}}, n^\delta)$-smooth densities.

The difference of our data structure with those in [1, 16] is in the absence of REP arrays. These arrays guarantee that when we move to a child of a node whose subtree contains $N$ nodes, then this child node will be the root of a subtree containing $\sqrt{N}$ nodes. In our case, this is not guaranteed (it is easy to come up with a setting where all elements are in a very small region and thus the height of our tree structure is large). However, assuming that the input elements are generated by a smooth distribution, it is *very unlikely* that this bad scenario will happen, since we prove that the height of our tree structure is doubly logarithmic with high probability. Our data structure is in a sense "similar" to other data structures that partition the space (e.g., quadtrees). Indeed, our structure partitions the universe until each region has a bounded number of elements. On the contrary, the use of REP arrays allows for a partition according to the number of elements (like e.g., in range trees), thus guaranteeing that each partition has geometrically less elements.

$O(1)$ **search time with high probability.** We study a random process of $rn$ insert (or delete) operations on this structure where in each operation, $j = 1, \ldots, rn$, with probability $p = (0, 1]$ a new element $X \in [a, b]$ obeying an unknown $(f_1(n), f_2(n)) = (\frac{n}{g(n)}, \ln^{O(1)} n)$-smooth distribution $\mu$, is inserted, otherwise a random existing key is deleted; here $g(n)$ denotes a function which is either constant or slowly growing with $n$ (i.e., $\ln^* n$). The class $(f_1(n), f_2(n)) = (\frac{n}{g(n)}, \ln^{O(1)} n)$-smooth distributions includes that of bounded ($(n, 1)$-smooth) densities, for which $O(1)$ expected search time was known [1], as well as all those for which a $o(\log \log n)$ expected search time could be achieved [1]; for instance, the density $\mu[0, 1](x) = - \ln x$ is $(n/(\log^* n)^{1+\epsilon}, \log^2 n)$-smooth, and an expected search time complexity of $\Theta(\log^* n)$ was given in [1]. Our result implies $O(1)$ search time with high probability for all the aforementioned densities.

The idea is as follows. We can prove (see [14]) that during each step $j = 1, \ldots, rn$, there are $O(n)$ elements stored. Then Theorem 1 establishes that during each step $j = 1, \ldots, rn$, *no bin* of the 1st Layer gets more than $\text{poly}\log n$ elements (balls), with high probability. That is, the whole tree-structure reduces to a single Layer. Since each $\text{BIN}(j_1)$, $j_1 = 1, \ldots, f_1(n)$, is implemented as a $q^*$-heap, we can search for element $y$ in it within $O(1)$ time. Finally, we can determine within O(1) time the bin $\text{BIN}(j_1)$ that $y$ may belong using the Expr. (3).

**Power Law Distributions.** As shown in Section 2, the efficiency of an arbitrary interpolation step dictating subtree $p$ highly relies on how the total of $n_p$ elements belonging to subtree $p$ are sparsely distributed in its associated subinterval $[a_p, b_p]$. This sparsity fails for power laws, as we show next. Let the *discrete* universe of possible keys be $U = \{1, 2, \ldots, N\}$, with $N$ arbitrarily large, spread over interval $I = [1, b]$ and listed in decreasing frequency. Each random key $X$ is drawn according to the power law distribution $\Pr[X \geq x] = cx^{-\beta}$ for constants $c, \beta > 0$ [17, Sec. 2]. The probability mass accumulated on subinterval $I_1 = [1, n^\alpha]$ containing the subset of keys $\{1, \ldots, n^\alpha\} \subseteq U$ equals:

$$\Pr[X \in I_1] = 1 - \Pr[X \geq n^\alpha] = 1 - \frac{c}{(n^\alpha)^\beta} = \omega(\frac{n^\delta}{n}),\ \delta < 1 \qquad (14)$$

which according to definition (2) means that subinterval $I_1$ is *not* $(f_1(n), f_2(n)) = (n^{\alpha}, n^{\delta})$-smooth for any constant $0 < \alpha < 1$. This rules out any attempt to employ IS on the whole interval $I = [1, b]$. However, $I_2 = I \setminus I_1$ can be arbitrarily sparse, as a function of $\alpha$, since $\Pr[X \in I_2] = \Pr[X \geq n^{\alpha}] = \frac{c}{(n^{\alpha})^{\beta}}$ and by setting $\alpha = \alpha(\beta) \geq \frac{1}{\beta}$, we get $\Pr[X \in I_2] = O(\frac{1}{n}) = O(\frac{f_2(n)}{n})$, with $f_2(n) = \mathit{poly} \log n$. That is, if we draw a random key $X \in [1, b]$ according to power law $\Pr[X \geq x] = cx^{-\beta}$, it will belong to an *arbitrary* subinterval of $I_2$ with probability $O(\frac{f_2(n)}{n}) \leq \frac{\mathit{poly} \log n}{n}$. The later implies that the power law distribution with parameters $c, \beta$, if restricted to $I_2$ remains $(f_1(n), \mathit{poly} \log n)$-smooth. Thus, if the target element $y \in I_2$, then Theorem 2 guarantees that IS on $I_2$ takes $O(1)$ search time with high probability. On the other hand, observe that the discrete subuniverse of $U$, which is spread in $[1, n^{\alpha(\beta)}]$, has cardinality $|\{1, \ldots, n^{\alpha(\beta)}\}| = O(n^{\alpha(\beta)})$. That is, if the target $y$ belongs to $I_1$, then it can be searched amongst $n^{\alpha(\beta)}$ possible keys, which is considerably smaller that the universe's cardinality $N = |U|$. Therefore, if $y \in I_1$, we can employ the van Emde Boas structure [24, 25], which yields a time complexity $O\left(\log \log \left(|\{1, \ldots, n^{\alpha(\beta)}\}|\right)\right) = O\left(\log \log \left(n^{\alpha(\beta)}\right)\right) = O(\log \log n)$. The splitting key $n^{\alpha(\beta)}$ yielding $\Pr[X \in I_2] = O(\frac{1}{n}) = O(\frac{f_2(n)}{n})$, with $f_2(n) = \mathit{poly} \log n$, can be approximated by a key $x^*$ during the initialization of the structure, without knowledge of the parameters $(c, \beta)$ (details are given in [14]).

**Binomial Distributions.** We can identify the dense subinterval $I_1 = [np - \Delta, np + \Delta] \subseteq [a, b]$ around the mean value $np$ for any binomial distribution $B(n, p)$. Notice that $|I_1| = 2\Delta$ and since we can safely set $\Delta = o(n)$, we can similarly apply a van Emde Boas structure on $I_1$. Taking advantage of the binomial sharp tail bounds, the remaining subinterval $I_2 = [a, b] \setminus I_1$ will remain sparse enough to apply IS. The rest of the details follow similarly to those for power law distributions.

**Acknowledgment.** We are indebted to Lefteris Kirousis for various helpful discussions.

## References

1. A. Andersson and C. Mattson. Dynamic Interpolation Search in $o(\log \log n)$ Time. In *Proc. 20th Coll. on Automata, Languages and Programming* – ICALP'93, LNCS Vol. 700 (Springer 1993), pp. 15-27.
2. A. Anderson and M. Thorup. Tight(er) Worst-case Bounds on Dynamic Searching and Priority Queues. In *Proc. 32nd ACM Symposium on Theory of Computing* – STOC 2001, pp.335-342. ACM, 2000.
3. F.W. Burton and G.N. Lewis. A robust variation of Interpolation Search. *Information Processing Latters*, 10, 198–201, 1980.
4. P. Beame and F. Fich. Optimal bounds for the predecessor problem and related problems. *Journal of Computer and System Sciences*, 65(1):38-72, 2002.
5. E. Demaine, T. Jones, and M. Patrascu. Interpolation Search for Non-Independent Data. In *Proc. 15th ACM-SIAM Symp. on Discrete Algorithms* – SODA 2004, pp. 522-523.

6. W. Feller. *An Introduction to Probability Theory and Its Applications* Vol. II, 2nd Edition, Wiley, New York 1971.
7. K.E. Foster. A statistically based interpolation binary search. TR, Winthrop College, SC.
8. G. Frederickson. Implicit Data Structures for the Dictionary Problem. *Journal of the ACM* 30(1):80-94, 1983.
9. G. Gonnet. Interpolation and Interpolation-Hash Searching. PhD Thesis. Waterloo: University of Waterloo 1977.
10. G. Gonnet, L. Rogers, and J. George. An Algorithmic and Complexity Analysis of Interpolation Search. *Acta Informatica* 13:39-52, 1980.
11. S. Huddleston and K. Mehlhorn. A new data structure for representing sorted lists. *Acta Informatica*, vol. 17 (1982), pp.157–184.
12. A. Itai, A. Konheim, and M. Rodeh. A Sparse Table Implementation of Priority Queues. In *Proc. 8th Coll. on Automata, Languages and Program.* – ICALP'81, LNCS Vol. 115 (Springer 1981), pp. 417-431.
13. A.C. Kaporis, C. Makris, S. Sioutas, A. Tsakalidis, K. Tsichlas and C. Zaroliagis. Improved bounds for finger search on a RAM. In *Algorithms* – ESA 2003, LNCS Vol. 2832 (Springer 2003), pp. 325-336.
14. A.C. Kaporis, C. Makris, S. Sioutas, A. Tsakalidis, K. Tsichlas and C. Zaroliagis. Dynamic Interpolation Search Revisited. Computer Technology Institute Tech. Report TR 2006/04/02, April 2006.
15. Y. Manolopoulos, Y. Theodoridis, V. Tsotras. *Advanced Database Indexing.* Kluwer Academic Publishers, 2000.
16. K. Mehlhorn and A. Tsakalidis. Dynamic Interpolation Search. *Journal of the ACM*, 40(3):621-634, July 1993.
17. M. Mitzenmacher. A Brief History of Generative Models for Power Law and Lognormal Distributions. *Internet Mathematics*, 1(2):226-251, 2004.
18. Y. Perl and L. Gabriel. Arithmetic Interpolation Search for Alphabet Tables. *IEEE Transactions on Computers*, 41(4):493-499, 1992.
19. Y. Perl, A. Itai, and H. Avni. Interpolation Search – A $\log\log N$ Search. *Communications of the ACM* 21(7):550-554, 1978.
20. Y. Perl, E. M. Reingold. Understanding the Complexity of the Interpolation Search. *Information Processing Letters* 6(6):219-222, December 1977.
21. W.W. Peterson. Addressing for Random Storage. *IBM Journal of Research and Development* 1(4):130-146, 1957.
22. N. Santorno and J.B. Sidney. Interpolation binary search. Information Processing Letters, 20, 179–181, 1985.
23. J. Spencer. Ten Lectures on The Probabilistic Method. Society for Industrial and Applied Mathematics, 2nd Ed. (1994)
24. P. van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. Information Processing Letters, 6:80-82, 1977.
25. P. van Emde Boas, R. Kaas, E. Zijlstra. Design and implementation of an efficient priority queue. Mathematical Systems Theory, 10:99-127, 1977.
26. D.E. Willard. Searching Unindexed and Nonuniformly Generated Files in $\log\log N$ Time. *SIAM Journal of Computing* 14(4):1013-1029, 1985.
27. D.E. Willard. Examining Computational Geometry, Van Emde Boas Trees, and Hashing from the Perspective of the Fusion Tree. *SIAM Journal on Computing*, 29(3), 1030-1049, 2000.
28. A.C. Yao and F.F. Yao. The Complexity of Searching an Ordered Random Table. In *Proc. 17th IEEE Symp. on Foundations of Computer Science* – FOCS'76, pp. 173-177, 1976.