The Cost of Cache-Oblivious Searching*

Michael A. Bender† Gerth Stølting Brodal‡ Rolf Fagerberg§ Dongdong Ge¶
Simai He‖ Haodong Hu** John Iacono†† Alejandro López-Ortiz‡‡

Abstract

This paper gives tight bounds on the cost of cache-oblivious searching. The paper shows that no cache-oblivious search structure can guarantee a search performance of fewer than $\lg e \log_B N$ memory transfers between any two levels of the memory hierarchy. This lower bound holds even if all of the block sizes are limited to be powers of 2. The paper gives modified versions of the van Emde Boas layout, where the expected number of memory transfers between any two levels of the memory hierarchy is arbitrarily close to $[\lg e + O(\lg \lg B / \lg B)] \log_B N + O(1)$. This factor approaches $\lg e \approx 1.443$ as B increases. The expectation is taken over the random placement in memory of the first element of the structure.

Because searching in the disk-access machine (DAM) model can be performed in $\log_B N + O(1)$ block transfers, this result establishes a separation between the (2-level) DAM model and cache-oblivious model. The DAM model naturally extends to k levels. The paper also shows that as k grows, the search costs of the optimal k-level DAM search structure and the optimal cache-oblivious search structure rapidly converge. This result demonstrates that for a multilevel memory hierarchy, a simple cache-oblivious structure almost replicates the performance of an optimal parameterized k-level DAM structure.

Keywords: cache-oblivious B-tree, cache-oblivious searching, van Emde Boas layout.

1 Introduction

Hierarchical Memory Models. Traditionally, algorithms are designed to run efficiently in a *random access model (RAM)* of computation, which assumes a flat memory with uniform access times. However, as hierarchical memory systems become steeper and more complicated, algorithms are increasingly designed assuming more accurate memory models; see e.g., [2–5, 7–9, 28, 33–35, 39–41]. Two of the most successful memory models are the *disk-access model (DAM)* and the *cache-oblivious model*.

*An earlier version of this paper in extended abstract form appears in the *Proceedings of the 44th Annual Symposium on the Foundations of Computer Science (FOCS)*, pages 271–280, 2003 [13].

†Department of Computer Science, Stony Brook University, Stony Brook, NY 11794-4400, USA, and Tokutek, Inc. Lexington, MA. http://www.tokutek.com. Email: bender@cs.sunysb.edu Supported in part by NSF Grants CCF 0621439/0621425, CCF 0540897/ 05414009, CCF 0634793/0632838, CNS 0627645, and CCF 0937822 and by DOE Grant DE-FG02-08ER25853.

‡MADALGO - Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation, Department of Computer Science, Aarhus University, Ny Munkegade, 8000 Århus C, Denmark. Email: gerth@cs.au.dk. Partially supported by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT) and the Carlsberg Foundation (contract number ANS-0257/20).

§Department of Mathematics and Computer Science, University of Southern Denmark, Campusvej 55, DK-5230 Odense M, Denmark. Email: rolf@imada.sdu.dk.

¶Department of Management Science and Engineering, Antai School of Economics and Management, Shanghai JiaoTong University, Shanghai, China, 200052. Email: ddge@sjtu.edu.cn

‖Department of System Engineering and Engineering Management, Chinese University of Hongkong, Hongkong, China. Email: smhe@se.cuhk.edu.hk.

**Networking and Device Connectivity at Windows Division, Microsoft, Redmond, WA, 98052. Email: haodhu@microsoft.com

††Department of Computer and Information Science, Polytechnic University, Brooklyn NY 11201, USA. Email: jiacono@poly.edu. Research supported in part by NSF grants CCF-0430849 and OISE-0334653 and by an Alfred P. Sloan Fellowship.

‡‡School of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada. Email: alopez-o@uwaterloo.ca.

The DAM model, developed by Aggarwal and Vitter [4], is a two-level memory model, in which the memory hierarchy consists of an internal memory of size M and an arbitrarily large external memory partitioned into blocks of size B. Algorithms are designed in the DAM model with full knowledge of the values of B and M. Because memory transfers are relatively slow, the performance metric is the number of block transfers.

The cache-oblivious model, developed by Frigo, Leiserson, Prokop, and Ramachandran [27, 31], allows programmers to reason about a two-level memory hierarchy but to prove results about an unknown multilevel memory hierarchy. As in the DAM model, the objective is to minimize the number of block transfers between two levels. The main idea of the cache-oblivious model is that by avoiding any memory-specific parametrization (such as the block sizes) the cache-oblivious algorithm has an asymptotically optimal number of memory transfers between all levels of an unknown, multilevel memory hierarchy.

Optimal cache-oblivious algorithms have memory performance (i.e., number of memory transfers) that is within a constant factor (independent of B and M) of the memory performance of the optimal DAM algorithm, which knows B and M. There exist surprisingly many (asymptotically) optimal cache-oblivious algorithms (see, e.g., [24, 30]).

I/O-Efficient Searching. This paper focuses on the fundamental problem of searching: Given a set S of N comparison-based totally-ordered elements, produce a data structure that can execute searches (or predecessor queries) on items in S.

We prove tight bounds on the cost of cache-oblivious searching. We show that no cache-oblivious search structure can guarantee that a search performs fewer than $\lg e \log_B N$ block transfers between any two levels of the memory hierarchy, even if all of the block sizes are limited to powers of 2.[1] We also give search structures in which the expected number of block transfers between any two levels of the memory hierarchy is arbitrarily close to $[\lg e + O(\lg \lg B / \lg B)] \log_B N + O(1)$, which approaches $\lg e \log_B N + O(1)$ for large B. This expectation is taken over the random placement in memory of the first element of the structure.

In contrast, the performance of the B-tree [10, 23], the classic optimal search tree in the DAM model, is as follows: A B-tree with N elements has nodes with fan-out B, which are designed to fit into one memory block. The B-tree has height $\log_B N + 1$, and a search requires $\log_B N + 1$ memory transfers.

A static cache-oblivious search tree, proposed by Prokop [31], also performs searches in $\Theta(\log_B N)$ memory transfers. The static cache-oblivious search tree is built as follows: Embed a complete binary tree with N nodes in memory, conceptually splitting the tree at half its height, thus obtaining $\Theta(\sqrt{N})$ subtrees each with $\Theta(\sqrt{N})$ nodes. Lay out each of these trees contiguously, storing each recursively in memory. This type of recursive layout is commonly referred to in the literature as a *van Emde Boas layout* because it is reminiscent of the recursive structure of the van Emde Boas tree [37, 38]. The static cache-oblivious search tree is a basic building block of most cache-oblivious search structures, including the (dynamic) cache-oblivious B-tree [14, 15, 15, 22, 32] and other cache-oblivious search structures [1, 6, 11, 12, 16–21, 25, 26]. Any improvements to the static cache-oblivious search structure immediately translate to improvements in these dynamic structures.

Results. We present the following results:

- We give an analysis of Prokop's static cache-oblivious search tree [31], proving that searches perform at most $2\left(1 + \frac{3}{\sqrt{B}}\right) \log_B N + O(1)$ expected memory transfers; the expectation is taken only over the random placement of the data structure in memory. This analysis is tight to within a $1 + o(1)$ factor.
- We then present a class of generalized van Emde Boas layouts that optimizes performance through the use of uneven splits on the height of the tree. For any constant $\varepsilon > 0$, we optimize the layout achieving a performance of $[\lg e + \varepsilon + O(\lg \lg B / \lg B)] \log_B N + O(1)$ expected memory transfers. As

[1] Throughout the paper $\lg N$ means $\log_2 N$.

before, the expectation is taken over the random placement of the data structure in memory[2]. While the derivations in the proof of the upper bound might not provide great insight into the reasons why it holds, its mere existence is valuable information. It shows that the tightness of the lower bound and points to the surprising fact that uneven splits, contrary to established practice, are of key importance in the building of this data structure.

- Intuitively, the improvement of uneven splitting, as compared to the even splitting in the standard van Emde Boas layout, is likely to be due to the generation of a variety of subtree sizes at each recursive level of the layout. Such a variety will on any search path reduce the number of subtrees that can have particularly bad sizes compared to the block size B. We strengthen this intuition by presenting another generalization of the van Emde Boas layout, which generates an explicit distribution of subtree sizes at each recursive level. The definition of this layout is more involved than for the first layout presented, but the corresponding proof of complexity is shorter and more direct. The achieved complexity is the same as for the first layout.

- Finally, we demonstrate that it is harder to search in the cache-oblivious model than in the DAM model. Previously the only lower bound for searching in the cache oblivious model was the $\log_B N$ lower bound from the DAM model. We prove a lower bound of $\lg e \log_B N$ memory transfers for searching in the average case in the cache-oblivious model. Thus, for large B, our upper bound is within a factor of $1 + o(1)$ of the optimal cache-oblivious layout.

Interpretation. We present cache-oblivious search structures that take 44% more block transfers than the optimal DAM structure, and we prove that one cannot do better. However, this result does not mean that our cache-oblivious structures are 44% slower than an optimal algorithm for a multilevel memory hierarchy. To the contrary, this worst-case behavior only occurs on a two-level memory hierarchy. To design a structure for a k-level memory hierarchy, one can extend the DAM model to k levels. A data structure for a k-DAM is designed with full knowledge of the size and block size of each level of the memory hierarchy. Thus, the 2-DAM is the standard DAM where searches cost $\log_B N + 1$ block transfers (using a B-tree). Surprisingly, in the 3-DAM this performance cannot be replicated in general. We show in Corollary 2.3 that a 3-DAM algorithm cannot achieve less than $1.207 \log_B N$ block transfers on all levels simultaneously. Thus, the performance gap between a 3-DAM and the optimal cache-oblivious structure is about half that of the 2-DAM and the optimal cache-oblivious structure; naturally, a modern memory hierarchy has more than three levels. Furthermore, we show that as the number k of levels in the memory hierarchy grows, the performance loss of our cache-oblivious structures relative to an optimal k-DAM structure tends to zero. Thus, for a modern memory hierarchy, our cache-oblivious structures combines simplicity and near-optimal performance.

Our cache-oblivious search trees also provide new insight into the optimal design strategy for divide-and-conquer algorithms. More generally, it has been known for several decades that divide-and-conquer algorithms frequently have good data locality [36]. The cache-oblivious model helps explain why divide-and-conquer can be advantageous.

When there is a choice, the splitting in a divide-and-conquer algorithm is traditionally done evenly. The unquestioned assumption is that splitting evenly is best. Our new search structures reveal a setting in which it is crucial that the resulting subtrees across the recursive decomposition not be of equal size.

Indeed, even splits lead to a performance slowdown factor of two in the worst case as each access might straddle two blocks. In contrast uneven splits lead to an expected slowdown of at most $\lg e \sim 1.443$. This shows that so long as the placement in memory is random, the tree accesses are reasonably well aligned with the cache blocks on the average. In this case, the intuitive reasons behind the superiority of uneven split is that the regularity of even splits allows for the construction of adversarial worst case configurations. This can be observed from the proof of Theorem 3.3.

[2]Note that explicit programmer intervention might be needed to ensure random placement of the data structure. This could be accomplished by means such as requesting a block of a fixed larger size and then placing the data structure at a small random offset from the head of the allocated block.

2 Lower Bound for Cache-Oblivious Searching

In this section we prove lower bounds on the I/O cost of comparison-based search algorithms optimizing for several block sizes at the same time. The specific problem we consider is the average cost of successful searches among N distinct elements, for a uniform distribution of the search key y on the N input elements. As average case lower bounds are also worst case lower bounds, our results also apply to the worst case cost. We emphasize that our bounds hold even if the block sizes are known to the algorithm, and that no randomization of the placement of the data structure of the search algorithm is assumed.

Formally, our model is as follows. Given a set S of N elements $x_1 < \cdots < x_N$ from a totally ordered universe, a *search structure* for S is an array M containing elements from S, possibly with several copies of each. A *search algorithm* for M is a binary decision tree where each internal node is labeled with either $y < M[i]$ or $y \leq M[i]$ for some array index i, and each leaf is labeled with a number $1 \leq j \leq N$. A search on a key y proceeds in a top-down fashion in the tree, and at each internal node advances to the left child if the comparison given by the label is true, otherwise it advances to the right. The binary decision tree is a *correct* search algorithm if for any $x_i \in S$, the path taken by a search on key $y = x_i$ ends in a leaf labeled i. Any such tree must have at least N leaves, and by pruning paths not taken by any search for $x_1, \ldots, x_N$, we may assume that it has exactly N leaves.

To add I/Os to the model, we divide the array M into contiguous *blocks* of size B. An internal node of a search algorithm is said to *access* the block containing the array index i in the label of the node. We define the *I/O cost of a search* to be the number of distinct blocks of M accessed on the path taken by the search.

We assume in our model that block sizes are powers of two and that blocks start at memory addresses divisible by the block size. This reflects the situation on actual machines, and entails no loss of generality, as any cache-oblivious algorithm at least should work for this case. The assumption implies that when considering two block sizes $B_1 < B_2$, a block of size B_1 is contained in exactly one block of size B_2.

The *depth* of a leaf in a tree is the number edges on its path to the root, and the *height* of a tree is the maximal depth among its leaves. We recall the following standard result on the average depth of leaves in binary trees.

Lemma 2.1 ([29, Section 2.3.4.5]) *For a binary tree with N leaves, the average depth of a leaf is at least $\lg N$.*

In our lower bound proof, we analyze the I/O cost of a given search algorithm with respect to several block sizes simultaneously. We first describe our method for the case of two block sizes. This leads to a lower bound of $1.207 \log_B N$ block transfers. We then generalize this proof to a larger number k of block sizes, and prove that in the limit as k grows, this gives a lower bound of $\lg e \log_B N \approx 1.443 \log_B N$ block transfers.

Lemma 2.2 *If a search algorithm on a search structure for block sizes B_1 and B_2, where $B_2 = {B_1}^c$ and $1 < c \leq 2$, guarantees that the average number of distinct blocks read during searches is at most $\delta \log_{B_1} N$ and $\delta \log_{B_2} N$, respectively, then*

$$\delta \geq \frac{1}{2/c + c - 2 + 3/(c \lg B_1)} .$$

Proof: Let T denote the binary decision tree constituting the search algorithm. The main idea of the proof is to transform T into a new binary decision tree T' by a transformation on nodes along each search path. Nodes that access a new block on the search path will be substituted by inserting small binary decision trees in their place. These decision trees are carefully chosen to have low height *and* to allow path nodes not accessing new blocks to be discarded. The end result is that the transformed path consists of one root-to-leaf path from each of the trees inserted along the original path, and there is one such tree for each distinct block access along the original path. By the bounded height of the inserted trees, a lower bound on the lengths of the transformed paths gives a lower bound on the distinct block accesses along the original paths. Lemma 2.1 supplies such a lower bound.

The transformation will be done simultaneously for all search paths in T by a top-down process, maintaining a decision tree at all times. During the process, it will be an invariant that a search for any key ends in a leaf having the same label as the leaf where the same search ends in T. In particular, the final tree T' is a correct search algorithm if T is.

To count the number of I/Os of each type (size B_1 blocks and size B_2 blocks) for each path in T, we mark some of the internal nodes by tokens τ_1 and τ_2. A node v is marked iff none of its ancestors accesses the size B_1 block accessed by v, i.e. if v is the first access to the block. The node v may also be the first access to the size B_2 block accessed by v. In this case, v is marked by τ_2, else it is marked by τ_1. Note that the word "first" above corresponds to viewing each path in the tree as a timeline—this view will be implicit in the rest of the proof.

For any root-to-leaf path, let b_i denote the number of distinct size B_i blocks accessed and let a_i denote the number of τ_i tokens on the path, for $i = 1, 2$. By the assumption stated above Lemma 2.1, a first access to a size B_2 block implies a first access to a size B_1 block, so we have $b_2 = a_2$ and $b_1 = a_1 + a_2$.

As said, we transform T into a new binary decision tree T' in a top-down fashion. The basic step in the transformation is to substitute a marked node v with a specific binary decision tree T_v resolving the order-wise relation between the search key y and a carefully chosen subset S_v of the elements stored in M. More precisely, in each step of the transformation, the subtree rooted at v is first removed, then the tree T_v is inserted at v's former position, and finally a copy of one of the two subtrees rooted at the children of v is inserted at each leaf of T_v. The set S_v is always chosen such that it includes the element accessed by the comparison in v. Hence, at each leaf of T_v, the order-wise relation to that element is known, and the appropriate one of the two subtrees can be inserted. The top-down transformation then continues downwards from the leaves of T_v. When the transformation reaches a leaf of T, it is left unchanged. Note that the invariant mentioned above is maintained at each step.

We now describe the tree T_v inserted, given that the set S_v of elements has been selected. The tree T_v is a binary decision tree of minimal height resolving the relation of the search key y to all keys in S_v. If we have $S_v = \{z_1, z_2, \ldots, z_t\}$, with elements listed in sorted order, this amounts to resolving to which of the $2t + 1$ intervals

$$(-\infty; z_1)\,,\ \ [z_1; z_1]\,,\ \ (z_1; z_2)\,, \ldots,\ [z_t; z_t]\,,\ \ (z_t; \infty)$$

y belongs. That tree has height at most $\lceil \lg(2t + 1) \rceil$, since a perfectly balanced binary search tree on S_v (having, say, comparisons of type "$<$" in the internal nodes), with one layer of t nodes added at the leaves to resolve the equality questions, will do.

We now describe how the set S_v is chosen. First consider the case of a node v marked τ_2. Here, we let the subset S_v consist of the at most B_1 distinct elements in the block of size B_1 accessed by v, plus every $\frac{B_2}{2B_1}$th element in sorted order among the at most B_2 distinct elements in the block of size B_2 accessed by v. The size of S_v is at most $B_1 + B_2/(B_2/(2B_1)) = 3B_1$, so the tree T_v has height at most $\lceil \lg(6B_1 + 1) \rceil$. As B_1 is a power of two, $\lg(8B_1)$ is an integer and hence an upper bound on the height.

Next, for the case of a node v marked τ_1, note that v in T has exactly one ancestor u marked τ_2 that accesses the same size B_2 block β as v does. When the tree T_u was substituted for u, the inclusion in S_u of the $2B_1$ evenly sampled elements from β ensures that below any leaf of T_u, at most $\frac{B_2}{2B_1} - 1$ of the elements in β can still have an unknown relation to the search key. We let S_v be these $\frac{B_2}{2B_1} - 1$ elements. The corresponding tree T_v has height at most at most $\lceil \lg(2(B_2/2B_1 - 1) + 1) \rceil$, which is at most $\lg B_2/B_1$, as B_1 and B_2 are powers of two.

For the case of an unmarked internal node v (i.e. a node where the size B_1 block accessed at the node has been accessed before), we can simply discard v together with either the left or right subtree, since we already have resolved the relation between the search key y and the element accessed at v. This follows from the choice of trees inserted at marked nodes: when we access a size B_2 block β_2 for the first time at some node u, we resolve the relation between the search key y and all elements in the size B_1 block β_1 accessed at u (due to the inclusion of all of β_1 in S_u), and when we the first time access a key in β_2 outside β_1, we resolve all remaining relations between y and elements in β_2.

By the height stated above for the inserted T_v trees, it follows that if a search for a key y in T corresponds to a path containing a_1 and a_2 tokens of type τ_1 and τ_2, respectively, then the search in T' corresponds to

a path with length bounded from above by the following expression.

$$a_2 \lg(8B_1) + a_1 \lg \frac{B_2}{B_1} = b_2 \lg(8B_1) + (b_1 - b_2) \lg \frac{B_2}{B_1} = b_2 \left[\lg(8B_1) - \lg \frac{B_2}{B_1} \right] + b_1 \lg \frac{B_2}{B_1}$$

The coefficients of b_2 and b_1 are positive by the assumption $B_1 < B_2 \leq {B_1}^2$, so upper bounds on b_1 and b_2 imply an upper bound on the expression above. By assumption, the average over all searches of b_1 and b_2 are bounded by $\delta \log_{B_1} N$ and $\delta \log_{B_2} N = (\delta \log_{B_1} N)/c$, respectively.

If we prune the tree for paths not taken by any search for the keys $x_1, \ldots, x_N$, the lengths of root-to-leaf paths can only decrease. The resulting tree has N leaves, and each leaf corresponds to a search, so averages over searches and averages over leaves are the same. As Lemma 2.1 gives a $\lg N$ lower bound on the average depth of a leaf, we get

$$\begin{aligned}
\lg N &\leq \frac{\delta}{c} \log_{B_1} N \left[\lg(8B_1) - \lg \frac{B_2}{B_1} \right] + \delta \log_{B_1} N \lg \frac{B_2}{B_1} \\
&= \frac{\delta}{c} \log_{B_1} N [3 + \lg B_1 - (c-1) \lg B_1] + \delta \log_{B_1} N (c-1) \lg B_1 \\
&= \delta \lg N [3/(c \lg B_1) + 1/c - (c-1)/c + (c-1)] \\
&= \delta \lg N [3/(c \lg B_1) + c + 2/c - 2] \, .
\end{aligned}$$

It follows that $\delta \geq 1/[3/(c \lg B_1) + c + 2/c - 2]$. □

Corollary 2.3 *If a search algorithm on a search structure guarantees, for all block sizes B, that the average number of distinct blocks read during searches is at most $\delta \log_B N$, then $\delta \geq 1/(2\sqrt{2} - 2) \approx 1.207$.*

Proof: Letting $c = \sqrt{2}$ in Lemma 2.2, we get $\delta \geq 1/[2\sqrt{2} - 2 + 3/(\sqrt{2} \lg B_1)]$. The lower bound follows by letting B_1 grow to infinity. □

Lemma 2.4 *If a search algorithm on a search structure for block sizes $B_1, B_2, \ldots, B_k$, where $B_i = {B_1}^{c_i}$ and $1 = c_1 < c_2 < \cdots < c_k \leq 2$, guarantees that the average number of distinct blocks read during searches is at most $\delta \log_{B_i} N$ for each block size B_i, then*

$$\delta \geq \frac{1}{\frac{2}{c_k} \left[1 + \frac{\lg(8k)}{2 \lg B_1} \right] + \sum_{i=1}^{k-1} \frac{c_{i+1}}{c_i} - k} \, .$$

Proof: The proof is a generalization of the proof of Lemma 2.2 for two block sizes, and we here assume familiarity with that proof. The transformation is basically the same, except that we have a token τ_i, $i = 1, \ldots, k$, for each of the k block sizes.

Again, a node v is marked if none of its ancestors access the size B_1 block accessed by v, i.e. if v is the first access to this block. The node v may also be the first access to blocks of larger sizes, and we mark v by τ_i, where B_i is the largest block size for which this is true. Note that by the assumption stated above Lemma 2.1, v must be the first access to the size B_j block accessed by v for all j with $1 \leq j \leq i$.

For any root-to-leaf path, let b_i denote the number of distinct size B_i blocks accessed and let a_i denote the number of τ_i tokens on the path, for $i = 1, \ldots, k$. We have $b_i = \sum_{j=i}^{k} a_j$. Solving for a_i, we get $a_k = b_k$ and $a_i = b_i - b_{i+1}$, for $i = 1, \ldots, k-1$.

As in the proof of Lemma 2.2, the transformation proceeds in a top-down fashion, and substitutes marked nodes v by binary decision trees T_v. We now describe the trees T_v for different types of nodes v.

For a node v marked τ_k, the tree T_v resolves the relation between the query key y and a set S_v of size $(2k-1)B_1$, consisting of the B_1 elements in the block of size B_1 accessed at v, plus for $i = 2, \ldots, k$ every $\frac{B_i}{2B_1}$th element in sorted order among the elements in the block of size B_i accessed at v. This tree can be chosen to have height at most $\lceil \lg(2(2k-1)B_1 + 1) \rceil \leq \lg(8kB_1)$.

For a node v marked τ_i, $i < k$, let β_j be the block of size B_j accessed by v, for $1 \leq j \leq k$. For $i+1 \leq j \leq k$, β_j has been accessed before, by the definition of τ_i. We now consider two cases. Case I is that β_{i+1} is the only block of size B_{i+1} that has been accessed inside β_k. By the definition of the tree T_u inserted at the ancestor u of v where β_k was first accessed, at most $B_{i+1}/2B_1 - 1$ of the elements in β_{i+1} can have unknown relations with respect to the search key y. The tree T_v inserted at v resolves these relations. It can be chosen to have height at most $\lg \frac{B_{i+1}}{B_1}$. Case II is that β_{i+1} is not the only block of size B_{i+1} that has been accessed inside β_k. Then consider the smallest j for which β_{j+1} *is* the only block of size B_{j+1} that has been accessed inside β_k (clearly, $j \leq k-1$). When we the first time accessed the second block of size B_j inside β_k at some ancestor u of v, this access was inside β_{j+1}, and a Case I substitution as described above took place. Hence a tree T_u was inserted which resolved all relations between the search key and elements in β_{j+1}. This includes the element accessed by the comparison at v, so the empty tree can be used for T_v, i.e. v and one of its subtrees is simply discarded.

For an unmarked node v, there is a token τ_i on the ancestor u of v in T for which the size B_1 block β_1 accessed by v was first accessed. This gave rise to a tree T_u in the transformation, and this tree resolved the relations between the search key and all elements in β_1, either directly ($i = k$) or by resolving the relations for all elements in a block containing β_1 ($1 \leq i < k$), so v and one of its subtrees can be discarded.

After transformation and final pruning, the length of a root-to-leaf path in the final tree is bounded from above by the following equation.

$$\begin{aligned}
& a_k \lg(8kB_1) + \sum_{i=1}^{k-1} a_i \lg \frac{B_{i+1}}{B_1} \\
= \;& b_k \lg(8kB_1) + \lg B_1 \sum_{i=1}^{k-1} (b_i - b_{i+1})(c_{i+1} - 1) \\
= \;& \lg B_1 \left[b_k \left(1 + \frac{\lg(8k)}{\lg B_1} \right) + b_1(c_2 - 1) \; + \sum_{i=2}^{k-1} b_i(c_{i+1} - c_i) - b_k(c_k - 1) \right] \\
= \;& \lg B_1 \left[b_k \left(2 + \frac{\lg(8k)}{\lg B_1} - c_k \right) + \sum_{i=1}^{k-1} b_i(c_{i+1} - c_i) \right].
\end{aligned}$$

For all i, the average value of b_i over all search paths is by assumption bounded by $\delta \log_{B_i} N = (\delta \log_{B_1} N)/c_i$, and the coefficient of b_i is positive, so we get the following upper bound on the average number of comparisons on a search path.

$$\begin{aligned}
& \delta \log_{B_1} N \lg B_1 \left[\frac{1}{c_k} \left(2 + \frac{\lg(8k)}{\lg B_1} - c_k \right) + \sum_{i=1}^{k-1} \frac{1}{c_i}(c_{i+1} - c_i) \right] \\
= \;& \delta \lg N \left[\frac{1}{c_k} \left(2 + \frac{\lg(8k)}{\lg B_1} \right) + \sum_{i=1}^{k-1} \frac{c_{i+1}}{c_i} - k \right].
\end{aligned}$$

By Lemma 2.1 we have

$$\delta \lg N \left[\frac{1}{c_k} \left(2 + \frac{\lg(8k)}{\lg B_1} \right) + \sum_{i=1}^{k-1} \frac{c_{i+1}}{c_i} - k \right] \geq \lg N \,,$$

and the lemma follows. □

Theorem 2.5 *If a search algorithm on a search structure guarantees, for all block sizes B, that the average number of distinct blocks read during searches is at most $\delta \log_B N$, then $\delta \geq \lg e \approx 1.443$.*

Proof: Let k be an integer, and for $i = 1, \ldots, k$ define $B_i = 2^{k+i-1}$. In particular, we have $B_1 = 2^k$ and $B_i = {B_1}^{c_i}$ with $c_i = (k+i-1)/k$. Consider the following subexpression of Lemma 2.4.

$$\begin{aligned}
& \frac{2}{c_k}\left(1+\frac{\lg(8k)}{2\lg B_1}\right)+\sum_{i=1}^{k-1}\frac{c_{i+1}}{c_i}-k \\
= \;& \frac{2k}{2k-1}\left(1+\frac{\lg(8k)}{2k}\right)+\sum_{i=1}^{k-1}\frac{k+i}{k+i-1}-k \\
= \;& \frac{2k}{2k-1}\left(1+\frac{\lg(8k)}{2k}\right)-1+\sum_{i=1}^{k-1}\frac{1}{k+i-1} \\
\le \;& \frac{2k}{2k-1}\left(1+\frac{\lg(8k)}{2k}\right)-1+\int_{k-1}^{2k-2}\frac{1}{x}\,dx \\
= \;& \frac{2k}{2k-1}\left(1+\frac{\lg(8k)}{2k}\right)-1+\ln 2.
\end{aligned}$$

Letting k grow to infinity Lemma 2.4 implies $\delta \ge 1/\ln 2 = \lg e$. □

3 Upper Bound for van Emde Boas Layout

In this section we give a tight analysis of the cost of searching in a binary tree stored using the van Emde Boas layout [31]. As mentioned earlier, in the vEB layout, the tree is split evenly by height, except for roundoff. Thus, a tree of height h is split into a top tree of height $\lceil h/2 \rceil$ and bottom tree of height $\lfloor h/2 \rfloor$. It is known [15,22] that the number of memory transfers for a search is $4\log_B N$ in the *worst case*; we give a matching configuration showing that this analysis is tight. We then consider the average-case performance over all starting positions of the tree in memory, and we show that the expected search cost is $2(1 + 3/\sqrt{B})\log_B N + O(1)$ memory transfers, which is tight within a $1 + o(1)$ factor. We assume that the data structure begins at a random position in memory; if there is not enough space, then the data structure "wraps around" to the first location in memory.

A relatively straightforward analysis of this layout shows that in the worst case the number of memory transfers is no greater than four times that of the optimal *cache-size-aware* layout. More formally,

Theorem 3.1 *Consider an $(N-1)$-node complete binary search tree that is stored using the Prokop vEB layout. A search in this tree has memory-transfer cost of at least $\left(4 - \frac{4}{1+\log_6 B}\right)\log_B N$ and at most $4\log_B N$ in the worst case.*

Proof: The upper bound has been established before in the literature [15,22]. For the lower bound we show that this value is achieved asymptotically. Let the block size be $B = \left(2^{2k}-1\right)/5$ for any even number $k \ge 4$ and consider a tree T of size $N-1$, where $N = 2^{k2^{m+1}}$ for some constant m. Number the positions within a block from 0 to $B-1$. As we recurse, we eventually obtain subtrees of size $5B = 2^{2k}-1$ and one level down of size $2^k - 1$. We align the subtree of size $5B$ that contains the root of T so that its first subtree R of size $2^k - 1$ (which also contains the root of T) starts in position $B-1$ of a block. In other words, any root-to-leaf search path in the subtree of size $5B$ crosses the block boundary because the root is in the last position of a block. Observe that R has 2^{k-1} subtrees of size $2^k - 1$ hanging from its bottom level. Number these $2^k - 1$ subtrees as they are laid in memory in the recursive decomposition and consider the $\lfloor 2(2^k+1)/5 \rfloor + 1$-th subtree of size $2^k - 1$ in this ordering. The root of this tree starts at a position between $B - 1 + (2^k - 1)(2(2^k+1)/5 - 1) + 1 = 3B - (2^k - 1)$ and $B - 1 + (2^k - 1)2(2^k+1)/5 = 3B - 1$. Thus, any root-to-leaf search path in this subtree crosses the block boundary. Observe that because trees are laid out consecutively, and $5B$ is a multiple of the block size, all other subtrees of size $5B$ start at position $B-1$ inside

a block and share the above that we can find a root-to-leaf path that has cost 4 inside this size-$5B$ subtree. Notice that a root-to-leaf path accesses 2^m many size-$5B$ subtrees, and if we choose the path according to the above position we know that the cost inside each size $5B$ subtree is 4, i.e., the first 4 blocks as a contiguous structure of size $5B$ may span at most 6 blocks. Thus, the total search cost is $4 \cdot 2^m$. Because $2^{k2^{m+1}} = N$ and $5B = 2^{2k} - 1$, we have

$$4 \cdot 2^m = \frac{4 \log_B N}{\log_B(5B+1)} = 4 \frac{\lg B}{\lg(5B+1)} \log_B N.$$

Furthermore, we bound the parameter $\lg B / \lg(5B+1)$ as follows:

$$\begin{aligned} \frac{\lg B}{\lg(5B+1)} &> \frac{\lg B}{\lg 6B} \\ &= 1 - \frac{\lg 6}{\lg 6 + \lg B} \\ &= 1 - \frac{1}{1 + \log_6 B}. \end{aligned}$$

Therefore, the total search cost has $4(1 - 1/(1+\log_6 B)) \log_B N$ memory transfers in the worst case. □

However, few paths in the tree have this property, which suggests that in practice, the Prokop vEB layout results in a much lower memory-transfer cost assuming random placement in memory.

In Theorem 3.3, appearing shortly, we formalize this notion. First, however, we give the following useful inequality to simplify the proof.

Claim 3.2 *Let B be a power of 2, t and t' be positive numbers satisfying $t/2 \le t' \le 2t$, $\sqrt{B}/2 \le t \le \sqrt{B}$, and $tt' \ge B$. Then*

$$2 + \frac{t+t'}{B} \le 2\left(1 + \frac{3}{\sqrt{B}}\right) \frac{\lg t + \lg t'}{\lg B}.$$

Proof: Because $t^2 + (t')^2 \le 5tt'/2$ for all $t/2 \le t' \le 2t$, we have

$$t + t' \le 3\sqrt{\frac{tt'}{2}}. \tag{1}$$

Define $x = tt'$ and define

$$f(x) = 2\left(1 + \frac{3}{\sqrt{B}}\right) \frac{\lg x}{\lg B} - 2 - \frac{3}{B}\sqrt{\frac{x}{2}}.$$

We will show that $f(x) \ge 0$ for $B \le x \le 2B$. First, we calculate the second derivative of $f(x)$.

$$f''(x) = -2\left(1 + \frac{3}{\sqrt{B}}\right) \frac{1}{x^2 \ln B} + \frac{3}{4\sqrt{2}\,B} \frac{1}{x^{3/2}}.$$

Because $x \le 2B$ (i.e., $x^{1/2} \le \sqrt{2B}$), we obtain

$$f''(x) \le \frac{1}{x^2}\left[\frac{3\sqrt{2B}}{4\sqrt{2}\,B} - 2\left(1 + \frac{3}{\sqrt{B}}\right) \frac{1}{\ln B}\right].$$

By removing the term $-6/(\sqrt{B} \ln B)$, we bound $f''(x)$ as follows:

$$f''(x) \le \frac{1}{x^2}\left(\frac{3}{4\sqrt{B}} - \frac{2}{\ln B}\right) \le 0.$$

Thus, we establish that $f(x)$ is convex in the range $B \le x \le 2B$. Because both $f(B)$ and $f(2B)$ are greater than zero, we obtain $f(x) \ge 0$ for $B \le x \le 2B$, which is equivalent to

$$2 + \frac{3}{B}\sqrt{\frac{x}{2}} \le 2\left(1 + \frac{3}{\sqrt{B}}\right)\frac{\lg x}{\lg B}.$$

From (1) and the above inequality, we obtain the follows:

$$2 + \frac{t + t'}{B} \le 2\left(1 + \frac{3}{\sqrt{B}}\right)\frac{\lg x}{\lg B}.$$

□

Theorem 3.3 *Consider a path in an $(N-1)$-node complete binary search tree of height h that is stored in vEB layout, with the start of its representation in memory determined uniformly at random within a block B. Then the expected memory-transfer cost of the search is at most $2(1 + 3/\sqrt{B}) \log_B N$.*

Proof: Although the recursion proceeds to the base case where trees have height 1, conceptually we stop the recursion at the level of detail where each recursive subtree has at most B nodes. We call those subtrees *critical recursive subtrees*, because they are recursive subtrees in the most "important" level of detail. Let the number of nodes in a subtree T be $|T|$. Therefore, any critical recursive subtree T has $|T|$ nodes, where $\sqrt{B}/2 \le |T| \le B$. Note that because of roundoff, we cannot guarantee that $|T| \ge \sqrt{B}$. In particular, if a tree has $B+1$ nodes and its height h' is odd, then the bottom trees have height $\lfloor h'/2 \rfloor$, and therefore contain roughly $\sqrt{B}/2$ nodes. Then there are exactly $|T| - 1$ initial positions for the upper tree that results in T being laid out across a block boundary. Similarly there are $B - |T| + 1$ positions in which the block does not cross a block boundary. Hence, the local expected cost of accessing T is

$$\frac{2(|T| - 1)}{B} + \frac{B - |T| + 1}{B} = 1 + \frac{|T| - 1}{B}.$$

Now we need two cases to deal with the roundoff. If $\sqrt{B}/2 \le |T| \le \sqrt{B}$ for the critical recursive subtree T, then we consider the next larger level of detail. There exists another critical recursive subtree T' immediately above T on the search path in this level of detail. Notice that $|T||T'| \ge B$. Because otherwise we would consider the coarser level of detail for our critical recursive subtree. Because we cut in the middle, we know that $2|T'| \ge |T| \ge |T'|/2$. From Claim 3.2 the expected cost of accessing T and T' is at most

$$1 + \frac{|T| - 1}{B} + 1 + \frac{|T'| - 1}{B} \le 2\left(1 + \frac{3}{\sqrt{B}}\right)\frac{\lg(|T||T'|)}{\lg B}.$$

For $\sqrt{B} \le |T| \le B$ for the critical recursive subtree T, we show that the cost of accessing T is less than $2(1 + 1/\sqrt{B}) \lg |T| / \lg B$. Define $f(x)$ as follows:

$$f(x) = 2\frac{\lg x}{\lg B}\left(1 + \frac{1}{\sqrt{B}}\right) - 1 - \frac{x - 1}{B}.$$

By calculating

$$f''(x) = -\frac{2}{x^2 \lg B}\left(1 + \frac{1}{\sqrt{B}}\right) \le 0,$$

we know $f(x)$ is convex. Because both $f(\sqrt{B})$ and $f(B)$ are greater than zero, we obtain $f(x) \ge 0$ for the entire range $\sqrt{B} \le x \le B$. Thus, considering $f(|T|)$, we obtain that the expected cost of accessing T is

$$1 + \frac{|T| - 1}{B} \le 2\left(1 + \frac{1}{\sqrt{B}}\right)\frac{\lg |T|}{\lg B}.$$

Combining the above arguments, we conclude that although the critical recursive subtrees on a search path may have different sizes, their expected memory-transfer cost is at most

$$\sum_T 2\left(1+\frac{3}{\sqrt{B}}\right)\frac{\lg|T|}{\lg B} = 2\left(1+\frac{3}{\sqrt{B}}\right)\log_B N.$$

This is a factor of $2(1+3/\sqrt{B})$ times the (optimal) performance of a B-tree. □

4 Upper Bounds for Cache Oblivious Searching

We give two cache-oblivious layouts for achieving our main upper bound. One layout, which we term the *generalized van Emde Boas layout* is very simply defined, but has a lengthy proof of complexity. The other layout, termed *multi-layer van Emde Boas layout* is slightly more involved, but allows a shorter proof.

The intuition behind both methods is to ensure an appropriate diversity of sizes of subtrees at each level of recursion of a van Emde Boas type layout. When looking at the level of recursion where the sizes of subtrees get below the block size B, these sizes will then be spread out in the interval $[\sqrt{B}, B]$. Under randomization of the starting position of the data structure in memory, smaller subtrees will be less likely to straddle a block boundary thus avoiding a second I/O for these trees during a search. On the other hand, smaller subtrees mean more such subtrees on a search path, hence more I/Os. The main question is what is the best possible balance between the two effects holding true for any value of the block size B.

In the multi-layer van Emde Boas layout, we specify an explicit diversity of sizes of subtrees at each level of the recursion, which we can prove achieves a balance between the two effects that entails a cache-oblivious search time asymptotically meeting our lower bound.

In the van Emde Boas layout, we split the tree height in a fixed, uneven way during the recursion of the van Emde Boas layout, and we are able to show that this simple scheme achieves the same result. Intuitively, the recursive uneven division provides a spread of subtree sizes with the same effect as our explicit scheme.

5 Upper Bound for the Generalized van Emde Boas Layout

We now propose and analyze a *generalized van Emde Boas layout* having a better search cost. In the original vEB layout, the top recursive subtree and the bottom recursive subtrees have the same height (except for roundoff). At first glance this even division would seem to yield the best memory-transfer cost. Surprisingly, we can improve the van Emde Boas layout by selecting different heights for the top and bottom subtrees.

The generalized vEB layout is as follows: Suppose the complete binary tree contains $N-1 = 2^h - 1$ nodes and has height $h = \lg N$. Let a and b be constants such that $0 < a < 1$ and $b = 1 - a$. Conceptually we split the tree at the edges below the nodes of depth $\lceil ah \rceil$. This splits the tree into a *top recursive subtree* of height $\lceil ah \rceil$, and $k = 2^{\lceil ah \rceil}$ *bottom recursive subtrees* of height $\lfloor bh \rfloor$. Thus, there are roughly N^a bottom recursive subtrees and each bottom recursive subtree contains roughly N^b nodes. We map the nodes of the tree into positions in the array by recursively laying out the subtrees contiguously in memory. The base case is reached when the trees have one node, as in the standard vEB layout.

We find the values of a and b that yield a layout whose memory-transfer cost is arbitrarily close to $[\lg e + O(\lg\lg B/\lg B)]\log_B N + O(1)$ for $a = 1/2 - \xi$ and large enough N. We focus our analysis on the first level of detail where recursive subtrees have size at most the block size B. In our analysis memory transfers can be classified in two types. There are $\mathcal{V}$ *path-length memory transfers*, which are caused by accessing different recursive subtrees in the level of detail of the analysis, and there are $\mathcal{C}$ *page-boundary memory transfers*, which are caused by a single recursive subtree in this level of detail straddling two consecutive blocks. It turns out that each of these components has the same general recursive expression and differs only in the base cases. The total number of memory transfers is at most $\mathcal{V} + \mathcal{C}$ by linearity of expectation.

The recurrence relation obtained contains rounded-off terms ($\lfloor\cdot\rfloor$ and $\lceil\cdot\rceil$) that are cumbersome to analyze. We show that if we ignore the roundoff operators, then the error term is small. We obtain a solution expressed

in terms of a power series of the roots of the characteristic polynomial of the recurrence. We show for both $\mathcal{V}$ and $\mathcal{C}$ that the largest root is unique and hence dominates all other roots, resulting in asymptotic expressions in terms of the dominant root.

Using these asymptotic expressions, we obtain the main result, namely a layout whose total cost is arbitrarily close to $[\lg e + O(\lg\lg B/\lg B)]\log_B N + O(1)$ as the split factor $a = 1/2 - \xi$ approaches 1/2 and for N large enough. This performance matches the lower bound from the Section 2 up to low-order terms.

Causes of Memory Transfers: Path-Length and Block-Boundary-Crossing Functions

We let $\mathcal{B}(x)$ denote the expected block cost of a search in a tree of height x. To begin, we explain the base case for the recurrence, when the entire tree is a critical recursive subtree. Recall that a *critical recursive subtree* is a recursive subtree of size less than B. If a critical recursive subtree crosses a block boundary, then the block cost is 2; otherwise the block cost is 1. As in the Theorem 3.3, the expected block cost of accessing a critical recursive subtree T of size $|T| = t - 1$ and height $x = \lg t$ is

$$1 + \frac{t-2}{B} = 1 + \frac{2^x - 2}{B}.$$

Thus, the base case is when $|T| < B$, which means that $t \le B$ and $1 \le x \le \lg B$.

We next give the recurrence for the block cost $\mathcal{B}(x)$ of a tree T of height x. By linearity of expectation, the expected block cost is at most that of the top recursive subtree plus the bottom recursive subtree, i.e.,

$$\mathcal{B}(x) \le \mathcal{B}(\lceil ax \rceil) + \mathcal{B}(\lfloor bx \rfloor),$$

for $x > \lg B$, [3] for $a + b = 1$, $0 < a \le b < 1$.

We decompose (an upper bound on) the cost of $\mathcal{B}(x)$ into two pieces. Let $\mathcal{V}(x)$ be the number of critical recursive subtrees visited along a root-to-leaf path ($\mathcal{V}$ stands for "vertical"), i.e.,

$$\mathcal{V}(x) = \begin{cases} \mathcal{V}(\lceil ax \rceil) + \mathcal{V}(\lfloor bx \rfloor), & x > \lg B; \\ 1, & 1 \le x \le \lg B. \end{cases} \tag{2}$$

Let $\mathcal{C}(x)$ be the expected number of critical recursive subtrees straddling block boundaries along the root-to-leaf path ($\mathcal{C}$ stands for "crossing"), i.e.,

$$\mathcal{C}(x) = \begin{cases} \mathcal{C}(\lceil ax \rceil) + \mathcal{C}(\lfloor bx \rfloor), & x > \lg B; \\ (2^x - 2)/B, & 1 \le x \le \lg B. \end{cases} \tag{3}$$

Observe that both $\mathcal{V}(x)$ and $\mathcal{C}(x)$ are monotonically increasing. By linearity of expectation, we obtain

$$\mathcal{B}(x) \le \mathcal{V}(x) + \mathcal{C}(x)$$

for all $x \ge \lg B$.

The recurrences for $\mathcal{V}(x)$ and $\mathcal{C}(x)$ are both of the form

$$\mathcal{F}(x) = \mathcal{F}(\lceil ax \rceil) + \mathcal{F}(\lfloor bx \rfloor).$$

As we will see, it is easier to analyze a recurrence of the form

$$\mathcal{G}(x) = \mathcal{G}(ax) + \mathcal{G}(bx),$$

where the roundoff error is removed. In the next few pages, we show that $\mathcal{F}(x)$ can be approximated by $\mathcal{G}(x)$ as x increases. Afterwards, we show how to calculate $\mathcal{G}(x)$.

[3] We cannot claim equality, i.e., that $\mathcal{B}(x) = \mathcal{B}(\lceil ax \rceil) + \mathcal{B}(\lfloor bx \rfloor)$, because the leaf node of the top recursive subtree and root node of a bottom recursive subtree can belong to the same block. Thus, an equal sign in the recurrence might double count one memory transfer.

Roundoff Error Is Small

We next show that as x increases, the difference between $\mathcal{F}(x)$ and $\mathcal{G}(x)$ can be bounded. To quantify the difference between $\mathcal{F}(x)$ and $\mathcal{G}(x)$ — see Theorem 5.4 — we use functions $\beta(x)$ and $\delta(x)$ defined recursively below:

Definition 5.1 *Let $a < \min\{1/2, 1 - 2/\lg B\}$. Define the recursive function $\beta(x)$ and $\delta(x)$ as follows:*

$$\beta(x) = \begin{cases} 0, & x \le \lg B; \\ \beta(ax+1)+1, & x > \lg B. \end{cases}$$

$$\delta(x) = \begin{cases} 1, & x \le \lg B; \\ \delta(ax+1)(1 + 2a^{\beta(x)-2}/\lg B), & x > \lg B. \end{cases}$$

The following lemma gives upper and lower bounds of $\beta(x)$.

Lemma 5.2 *For all $x > \lg B$, the function $\beta(x)$ satisfies*

$$\frac{2}{a^2 x} \ge \frac{a^{\beta(x)-2}}{\lg B} \ge \frac{1}{2ax}. \tag{4}$$

Proof: For parameter n, define the *nth interval* I_n to be

$$I_n = \left[\frac{\lg B}{2a^{n-1}} + \frac{1}{1-a}, \frac{\lg B - 1 - a - \cdots - a^{n-1}}{a^n} \right].$$

We now prove the following inequality for all $x > \lg B$:

$$\frac{1}{2} \lg B \left(\frac{1}{a}\right)^{\beta(x)-1} \le x - \frac{1}{1-a} \le \lg B \left(\frac{1}{a}\right)^{\beta(x)}. \tag{5}$$

We establish (5) in two parts.

1. We first show that the inequality holds for all n and all $x \in I_n$.
2. We then explain that the interval $I_0 \cup I_1 \cup I_2 \cup \cdots$ covers the interval $[\lg B, \infty)$.

We now prove the first part, showing by induction on n that (5) holds for all n and all $x \in I_n$.
Base Case: The base case is when

$$x \in I_0 = \left[\frac{a}{2} \lg B + \frac{1}{1-a}, \lg B \right].$$

Because $a < 1/2$,

$$\frac{1}{1-a} > 0.$$

Therefore, because $x \in I_0$,

$$\frac{a}{2} \lg B \le x - \frac{1}{1-a} \le \lg B. \tag{6}$$

Because $x \le \lg B$ and from Definition 5.1, $\beta(x) = 0$. Observe that (6) is equivalent to (5) when $\beta(x) = 0$. Therefore, (5) holds in the base case.

Induction step: Assume that (5) holds for the nth interval I_n. We will show that (5) also holds for the $(n+1)$st interval I_{n+1}, i.e., when

$$x \in I_{n+1} = \left[\frac{\lg B}{2a^n} + \frac{1}{1-a}, \frac{\lg B - 1 - a - \cdots - a^n}{a^{n+1}} \right],$$

or equivalently when

$$\frac{\lg B}{2a^n} + \frac{1}{1-a} \leq x \leq \frac{\lg B - 1 - a - \cdots - a^n}{a^{n+1}}. \tag{7}$$

Multiplying by a and adding 1 to both sides of (7), we see that (7) is equivalent to

$$\frac{\lg B}{2a^{n-1}} + \frac{1}{1-a} \leq ax + 1 \leq \frac{\lg B - 1 - a - \cdots - a^{n-1}}{a^n},$$

i.e.,

$$ax + 1 \in I_n.$$

Thus, by induction (plugging $ax+1$ for x in (5)), we obtain

$$\frac{1}{2}\lg B \left(\frac{1}{a}\right)^{\beta(ax+1)-1} \leq ax + 1 - \frac{1}{1-a} \leq \lg B \left(\frac{1}{a}\right)^{\beta(ax+1)}.$$

Noticing that $\beta(ax+1) = \beta(x) - 1$ by Definition 5.1 and

$$(ax+1) - \frac{1}{1-a} = a\left(x - \frac{1}{1-a}\right),$$

we establish

$$\frac{1}{2}\lg B \left(\frac{1}{a}\right)^{\beta(x)-2} \leq a\left(x - \frac{1}{1-a}\right) \leq \lg B \left(\frac{1}{a}\right)^{\beta(x)-1},$$

which is equivalent to (5) for $x \in I_{n+1}$.

We now prove the second part, that $\bigcup_{n=0}^{\infty} I_n$ covers the interval $[\lg B, \infty)$. This claim follows when $a < 1 - 2/\lg B$, which is guaranteed when $B > 16$. The claim follows because intervals overlap, i.e., the right endpoint of the I_n is between the left and right endpoints of the I_{n+1}, that is,

$$\frac{\lg B}{2a^n} + \frac{1}{1-a} \leq \frac{\lg B - 1 - a - \cdots - a^{n-1}}{a^n} \leq \frac{\lg B - 1 - a - \cdots - a^n}{a^{n+1}}.$$

We have now established that (5) holds for all $x > \lg B$.

We next show that (5) is equivalent to the lemma statement, i.e., (4). Taking inverses on both sides of (5), we have

$$2\frac{a^{\beta(x)-1}}{\lg B} \geq \frac{1}{x - \frac{1}{1-a}} \geq \frac{a^{\beta(x)}}{\lg B}$$

i.e.,

$$\frac{1}{a^2 x - \frac{a^2}{1-a}} \geq \frac{a^{\beta(x)-2}}{\lg B} \geq \frac{1}{2ax - \frac{2a}{1-a}}.$$

Because $x > \lg B$ and $a < 1 - 2/\lg B$, we have $x > 2/(1-a)$, i.e., $a^2 x/2 > a^2/(1-a)$. Therefore, the left side of the above inequality is less than $2/(a^2 x)$. The right side is greater than $1/(2ax)$ because $2a/(1-a) > 0$. Thus, we prove the following

$$\frac{2}{a^2 x} \geq \frac{a^{\beta(x)-2}}{\lg B} \geq \frac{1}{2ax}$$

for all $x > \lg B$ as claimed. □

The following lemma gives the properties and the upper bound of $\delta(x)$.

Lemma 5.3 *The function $\delta(x)$ has the following properties:*

(1) If $\beta(x) = \beta(y)$, then $\delta(x) = \delta(y)$.

(2) For all $x > \lg B$,

$$(ax+1)\delta(ax+1) \leq ax\delta(x).$$

(3) For all $x > \lg B$,

$$\delta(x) \leq \exp\left[\frac{2}{a(1-a)\lg B}\right],$$

which is

$$1 + O\left(\frac{2}{a(1-a)\lg B}\right) = 1 + O\left(\frac{1}{\lg B}\right).$$

Proof: (1) This claim follows from Definition 5.1.
(2) This claim follows from Definition 5.1 of $\delta(x)$ and Lemma 5.2

$$\frac{a^{\beta(x)-2}}{\lg B} \geq \frac{1}{2ax}.$$

(3) Recall that from Definition 5.1, we have

$$\frac{\delta(x)}{\delta(ax+1)} = 1 + 2\frac{a^{\beta(x)-2}}{\lg B}$$

for all $x > \lg B$. Furthermore, because $1 + y < e^y$ is true for any $y > 0$, we bound the function $\delta(\cdot)$ as follows

$$\frac{\delta(x)}{\delta(ax+1)} \leq \exp\left[2\frac{a^{\beta(x)-2}}{\lg B}\right]. \tag{8}$$

For simplification, we define P_i be the polynomial $a^i x + a^{i-1} + \cdots + 1$. In the following, we show there exists some big integer n such that $P_{n+1} = a^{n+1}x + a^n + \cdots + 1 < \lg B$. First of all, because $a < 1$, a^n is arbitrary small when n goes to infinity. Thus, if n is big enough, then

$$a^{n+1}x < \frac{\lg B}{2} \tag{9}$$

for fixed number x. Second, for big $B > 16$, we have $1/(1-a) < (\lg B)/2$. Thus,

$$a^n + \cdots + a + 1 < \frac{1}{1-a} < \frac{\lg B}{2} \tag{10}$$

is true for all integer n. Therefore, combining both (9) and (10), we obtain that there exists some big integer n such that

$$P_{n+1} = a^{n+1}x + a^n + \cdots + 1 < \lg B,$$

which means, by Definition 5.1 of $\delta(x)$, $\delta(P_{n+1}) = \delta(a^{n+1}x + a^n + \cdots + 1) = 1$. Therefore, $\delta(x)$ can be expressed as the multiplication of $n+1$ items, i.e.,

$$\delta(x) = \frac{\delta(x)}{\delta(ax+1)}\frac{\delta(ax+1)}{\delta(a^2x+ax+1)}\cdots\frac{\delta(a^nx+a^{n-1}+\cdots+1)}{\delta(a^{n+1}x+a^n+\cdots+1)}$$

Using the term P_i in the above equation, we get the simplified version

$$\delta(x) = \prod_{i=0}^{n}\frac{\delta(P_i)}{\delta(P_{i+1})}. \tag{11}$$

To bound $\delta(x)$, we give the upper bound for $\delta(P_i)/\delta(P_{i+1})$ first. Notice that $P_{i+1} = aP_i + 1$, Replacing x by P_i in (8), we have the upper bound

$$\frac{\delta(P_i)}{\delta(P_{i+1})} \leq \exp\left[2\frac{a^{\beta(P_i)-2}}{\lg B}\right].$$

We claim that $\beta(P_i) = n + 1 - i$ for all $0 \leq i \leq n+1$. We prove this claim by induction. The base case is for P_{n+1}. From Definition 5.1 and $P_{n+1} < \lg B$, we have $\beta(P_{n+1}) = 0$. Assume the claim holds for some P_i. We prove the claim holds for P_{i-1}. Because $P_i = aP_{i-1} + 1$, we have $\beta(P_{i-1}) = \beta(P_i) + 1$ from Definition 5.1 of $\beta(x)$. Therefore, by induction, we obtain $\beta(P_{i-1}) = \beta(P_i) + 1 = n + 1 - i + 1 = n + 1 - (i-1)$ as claimed. Thus, each of those items $\delta(P_i)/\delta(P_{i+1})$ has the upper bound

$$\exp\left[2\frac{a^{n-i-1}}{\lg B}\right].$$

Therefore, we obtain

$$\delta(x) \leq \exp\left[2\sum_{i=0}^{n}\frac{a^{n-i-1}}{\lg B}\right] = \exp\left[\frac{2}{\lg B}\sum_{i=0}^{n}a^{i-1}\right].$$

Because

$$\sum_{i=0}^{n}a^{i-1} < \sum_{i=0}^{\infty}a^{i-1} = \frac{1}{a(1-a)},$$

we prove that

$$\delta(x) \leq \exp\left[\frac{2}{a(1-a)\lg B}\right],$$

as claimed. □

Theorem 5.4 (Roundoff Error) *Let $\mathcal{F}(x) = \mathcal{F}(\lceil ax \rceil) + \mathcal{F}(\lfloor bx \rfloor)$ and $\mathcal{G}(x) = \mathcal{G}(ax) + \mathcal{G}(bx)$ for $0 < a \leq b < 1$ and $a + b = 1$. Then for $B > 8$, all $x > 1$, and constant c,*

$$\mathcal{F}(x) \leq \mathcal{G}(x\,\delta(x)) \leq c\left[1 + O\left(\frac{1}{\lg B}\right)\right]x + O(1).$$

Proof: First recall that $\mathcal{F}(x)$ and $\mathcal{G}(x)$ are monotonically increasing. Thus, from $\lceil ax \rceil \leq ax + 1$ and $\lfloor bx \rfloor \leq bx$, we have

$$\mathcal{F}(x) \leq \mathcal{F}(ax + 1) + \mathcal{F}(bx). \tag{12}$$

We prove $\mathcal{F}(x) \leq \mathcal{G}(x\,\delta(x))$ inductively. The base case is when $1 < x \leq \lg B$, where $\delta(x) = 1$ from Definition 5.1 and $\mathcal{F}(x) = \mathcal{G}(x)$. Thus, $\mathcal{F}(x) \leq \mathcal{G}(x\,\delta(x))$ is true when $1 < x \leq \lg B$.

Assuming $\mathcal{F}(x) \leq \mathcal{G}(x\delta(x))$ is true for $1 < x \leq t$, we prove it is true for $1 < x \leq (t-1)/b$. Noticing that $(t-1)/b \leq \min\{t/b, (t-1)/a\}$ (because $b \geq a$), we have $ax + 1 \leq t$ and $bx \leq t$ for all $1 < x \leq (t-1)/b$. Thus, by assumption and (12), we obtain

$$\mathcal{F}(x) \leq \mathcal{G}((ax+1)\delta(ax+1)) + \mathcal{G}(bx\delta(bx)), \qquad 1 < x \leq (t-1)/b. \tag{13}$$

From Condition (2) in Lemma 5.3 and $\delta(bx) \leq \delta(x)$, we obtain

$$\mathcal{G}((ax+1)\delta(ax+1)) \leq \mathcal{G}(ax\delta(x)) \quad \text{and} \quad \mathcal{G}(bx\delta(bx)) \leq \mathcal{G}(bx\delta(x)). \tag{14}$$

Plugging (14) into (13), we derive that

$$\mathcal{F}(x) \leq \mathcal{G}(ax\delta(x)) + \mathcal{G}(bx\delta(x)) = \mathcal{G}(x\delta(x)), \qquad 1 < x \leq (t-1)/b.$$

Therefore, after two inductive steps, it is true for

$$1 < x \le \frac{t-1-b}{b^2},$$

and after n inductive steps, it is true for all

$$1 < x \le \frac{t-1-b-\cdots-b^{n-1}}{b^n} = \frac{t-(1-b^n)/(1-b)}{b^n}.$$

Therefore, as long as $t > 1/(1-b) = 1/a$, we have $\mathcal{F}(x) \le \mathcal{G}(x\delta(x))$ for all $x > 1$. Thus, we need $\lg B > 1/a$, which holds when $B > 8$ and $a > 1/3$.

Furthermore, if $\mathcal{G}(x) \le c\,x + O(1)$, then by Condition (3) in Lemma 5.3, we obtain the following:

$$\mathcal{F}(x) \le \mathcal{G}(x\delta(x)) \le c\,x\,\delta(x) + O(1) \le c[\,1 + O(1/\lg B)\,]x + O(1).$$

□

Bounding the Path-Length Function

We now determine the constant in the search cost $O(\log_B N)$, for given values of a and b. To do so, we assume

$$a = \frac{1}{q^k} \quad \text{and} \quad b = \frac{1}{q^m}\,, \tag{15}$$

for positive real number $q > 1$ and relatively prime integers m and k. Plugging (15) into $a+b=1$, we obtain $1/q^k + 1/q^m = 1$. Define

$$n = k - m\,. \tag{16}$$

Observe that because $k > m$ (since $a < b$), n is positive. We now have the simplified formula

$$q^k = q^n + 1\,. \tag{17}$$

The rationale behind this assumption is that this additional structure helps us in the analysis while still being dense; that is, for any given a and b satisfying $a+b=1$, we can find a' and b' defined as (15) that are arbitrary close to a and b. Because there exits a real number r such that $a = b^r$, we choose rational number k/m, $(k,m) = 1$ as close as desired to r. Let $q = b^{-1/m}$. Then $a' = 1/q^k$ and $b' = 1/q^m$. We call such an (a,b) pair a *twin power pair*.

As before we analyze $\mathcal{V}(x)$ first. We ignore the roundoff based on Theorem 5.4. Furthermore, we normalize the range for which $\mathcal{V}(x) = 1$ by introducing a function

$$H(x) = \begin{cases} H(ax) + H(bx)\,, & x > 1; \\ 1\,, & 0 < x \le 1. \end{cases} \tag{18}$$

Note that $\mathcal{V}(x \lg B) \le H(x\delta(x \lg B))$ by Theorem 5.4.

First we state a primary lemma of the subsection, which we prove later.

Lemma 5.5 *Let $(1/q^k, 1/q^m)$ be a twin power pair, and let $n = k - m$. Then for any constant $\varepsilon > 0$ and*

$$c_1 = \left(\sum_{i=1}^{n} q^{-i} + \sum_{i=n+1}^{k} q^{k-i}\right) / \left(kq^{k-1} - nq^{n-1}\right),$$

when $x \ge O(k/\varepsilon)$ we have

$$H(x) \le (c_1 + \varepsilon)q^k x + O(1).$$

Corollary 5.6 *For any constant $\varepsilon > 0$, the number $\mathcal{V}(x)$ of recursive subtrees on a root-to-leaf path is bounded by*

$$(c_1 + \varepsilon)q^k \log_B N + O(1),$$

when $N \geq B^{O(k/\varepsilon)}$.

We obtain the main upper-bound result of the paper by showing that $c_1 q^k \approx \lg e$ for some twin power pair.

Theorem 5.7 (Path-Length Cost) *For any constant $\varepsilon > 0$, the number of recursive subtrees on a root-to-leaf path is*

$$(\lg e + \varepsilon) \log_B N + O(1) \approx 1.443 \log_B N + O(1),$$

as the split factor $a = 1/2 - \xi$ approaches $1/2$.

Proof: Choose the twin power pair $a = 1/q^k$ and $b = 1/q^{k-1}$ such that

$$\frac{1}{q^k} + \frac{1}{q^{k-1}} = 1,$$

which is equivalent to

$$q^k = q + 1.$$

The approximate solution for the above equation is

$$q \approx 1 + \frac{\ln 2}{k},$$

for $k \to \infty$. Therefore, we have

$$a = \frac{1}{1+q} \approx \frac{1}{2 + \ln 2/k}. \tag{19}$$

From Lemma 5.5, for $m = k - 1$ (and therefore $n = 1$), we have

$$c_1 = \left(q^{-1} + \sum_{i=2}^{k} q^{k-i} \right) / \left(kq^{k-1} - 1 \right) = \frac{q^k - 1}{(q-1)(kq^k - q)} .$$

Thus, for large k, we obtain

$$c_1 q^k = \frac{q^k - 1}{q - 1} \frac{1}{k - \frac{1}{q^{k-1}}} \xrightarrow{k \to \infty} \frac{1}{\ln 2} = \lg e.$$

That is, for a given $\varepsilon/2 > 0$, we can choose a big constant k_ε such that

$$c_1 q^k \leq \lg e + \varepsilon/2, \tag{20}$$

for all $k \geq k_\varepsilon$.

From Corollary 5.6, for a given $\varepsilon/8 > 0$ and the above constant k_ε, we can choose big constant $N_{\varepsilon,k}$ such that

$$\mathcal{V}(x) \leq (c_1 + \varepsilon/8)q^k \log_B N + O(1), \tag{21}$$

for all $N \geq N_{\varepsilon,k}$. Plugging (20) into (21) and noticing that $q^k = 1/a < 4$, we obtain

$$\mathcal{V}(x) \leq (\lg e + \varepsilon) \log_B N + O(1) \approx 1.443 \log_B N + O(1)$$

as claimed.

Noticing that for big $k \geq k_\varepsilon$, we see that the split factor a approaches $1/2$ by (19). In particular, as long as

$$\xi \leq \frac{1}{2} - \frac{1}{2 + \ln 2/k_\varepsilon} = \frac{\ln 2}{4k_\varepsilon + \ln 4},$$

it suffices that the split factor $a = 1/2 - \xi$. □

To complete the proof of Lemma 5.5, we establish some properties of $H(x)$. Since $H(x)$ is monotonically increasing, we can bound the value $H(x)/x$ for $q^i \leq x \leq q^{i+1}$ as follows:

$$\frac{H(q^i)}{q^{i+1}} \leq \frac{H(x)}{x} \leq \frac{H(q^{i+1})}{q^i}. \tag{22}$$

Let H_{min} be the lower bound and H_{max} be the upper bound of $H(q^i)/q^i$, when i is larger than a given integer j. Noticing that the left part in Inequality (22) is $H(q^i)/q^{i+1} \geq H_{min}/q$ and the right part in Inequality (22) is $H(q^{i+1})/q^i \leq qH_{max}$, we obtain

$$\frac{H_{min}}{q} \leq \frac{H(x)}{x} \leq qH_{max},$$

when x is bigger than q^j.

We give the recurrence of $H(\cdot)$. From (18), we have that for $i \geq 0$,

$$H(q^{i+1}) = H(aq^{i+1}) + H(bq^{i+1}). \tag{23}$$

Plugging (15) into (23) and since $n = k - m$, we obtain

$$H(q^{i+1}) = H(q^{i-k+1}) + H(q^{i+n-k+1}). \tag{24}$$

For the sake of notational simplicity, we denote $\alpha_i = H(q^{i-k+1})$. Therefore, (24) is equivalent to

$$\alpha_{i+k} = \alpha_{i+n} + \alpha_i\,. \tag{25}$$

We define the characteristic polynomial function of Recurrence (25) as $w(x) = x^k - x^n - 1$. Let $r_1, r_2, \ldots, r_k$ be the (possibly complex) roots of $w(x)$. We claim below that these roots are all unique.

The following four lemmas supply basic mathematical knowledge behind the proof of Lemma 5.5.

Lemma 5.8 *The k roots of $w(x) = x^k - x^n - 1$ are unique, when k and n are relatively prime integers such that $1 \leq n < k$.*

Proof: We prove this lemma by contradiction. If a root r of $w(x)$ is not unique, then $(x - r)^2$ is a factor of $w(x)$, and $x - r$ is a factor of $w'(x) = kx^{n-1}(x^{k-n} - n/k)$. Thus, r is either 0 or a root of $x^{k-n} - n/k$. But 0 is not a root of $w(x)$. Therefore,

$$r^{k-n} = n/k, \tag{26}$$

which means $|r| < 1$ (because $n < k$).

On the other hand, because r is a root of $w(x)$, $w(r) = r^n(r^{k-n} - 1) - 1 = 0$. Plugging (26) into $w(r) = 0$, we obtain $r^n = k/(n - k)$, which means $|r| > 1$ (because $|k| > |k - n|$). This is the contradiction. Therefore, every root of $w(x)$ is unique. □

Because $w'(x) = kx^{k-1} - nx^{n-1} > 0$ when $x > 1$ and q is a root of $w(x)$ greater than 1 (see Equation (17)), there is one unique real root $q > 1$ of $w(x)$. Without loss of generality, let $r_1 = q$.

We now show that if the k roots of the characteristic polynomial function of a series are unique, then the series in question is a linear combination of power series $\{r_j^i\}$ of the roots.

Lemma 5.9 *Consider a series $\{\alpha_i\}$ satisfying $\alpha_{k+s} = \sum_{i=0}^{k-1} d_i \alpha_{i+s}$ for complex numbers d_i and any integer s, and let $r_1, r_2, \ldots, r_k$ be the k unique roots of the characteristic function $g(x) = x^k - \sum_{i=0}^{k-1} d_i x^i$ for the series $\{\alpha_i\}$. Then there exists complex numbers $c_1, c_2, \ldots, c_k$ such that for all i,*

$$\alpha_i = \sum_{j=1}^{k} c_j r_j^i. \tag{27}$$

Proof: First we show that we can find $c_1, c_2, \ldots, c_k$ such that for the base values of α_i, $\alpha_i = \sum_{j=1}^{k} c_j r_j^i$ for all $i = 0, \ldots, k-1$. This can be derived by observing that the determinant of the Vandermonde matrix

$$V = \begin{pmatrix} 1 & r_1 & \cdots & r_1^{k-1} \\ 1 & r_2 & \cdots & r_2^{k-1} \\ \cdots & \cdots & \cdots & \cdots \\ 1 & r_k & \cdots & r_k^{k-1} \end{pmatrix}$$

is nonzero, and that $c_1, c_2, \ldots, c_k$ are the solution of the system of linear equations

$$(\alpha_0, \alpha_1, \ldots, \alpha_{k-1}) = (c_1, c_2, \ldots, c_k)V.$$

Now we show that for all $i \geq 0$,

$$\alpha_i = \sum_{j=1}^{k} c_j r_j^i.$$

Define

$$\beta_i = \sum_{j=1}^{k} c_j r_j^i \,.$$

We show that $\{\alpha_i\}$ and $\{\beta_i\}$ are the same recursive series. We know that $\beta_i = \alpha_i$ when $0 \leq i \leq k-1$. Because $r_1, r_2, \ldots, r_k$ are the k unique roots of the characteristic function $g(x)$, we know that the power series $\{r_j^i\}$ satisfies the same recursive formula as $\{\alpha_i\}$. Thus $\{\beta_i\}$ satisfies the same recursive formula (for all $s \geq 0$, $b_{k+s} = \sum_{i=0}^{k-1} d_i b_{i+s}$) by linearity. Now observe that the k base values together with the inductive formula uniquely determine the series and hence $\alpha_i = \beta_i$ for all $i \geq 0$. □

Hence we can solve Recurrence (25) by finding c_i that satisfy $\alpha_i = \sum_{j=1}^{k} c_j r_j^i$ for $i = 0, \ldots, k-1$. The base cases of $\{\alpha_i\}_{i=0}^{k-1}$ are determined by the original definition of $\alpha_i = H(q^{i-k+1})$. Because $0 < q^{i-k+1} < 1$ for $i = 0, \ldots, k-1$, we obtain $H(q^{i-k+1}) = \alpha_i = 1$.

Lemma 5.10 *The dominant root (i.e., the root with the largest absolute value) for $w(x) = x^k - x^n - 1$ is $r_1 = q$. All other roots r_2,...r_k have absolute value less than q.*

Proof: We first show that all other roots have magnitude less than q. Suppose that the magnitude of a root r_j (other than r_1) is $|r_j| = d$. We show that $d \leq q$. Since r_j is a root we have

$$1 = |(r_j^{k-n} - 1)\, r_j^n| = |r_j^{k-n} - 1||r_j^n| \geq (|r_j^{k-n}| - 1)\, d^n = d^k - d^n, \tag{28}$$

which means $w(d) = d^k - d^n - 1 \leq 0$. Because $w(q) = 0$ and $w'(x) = kx^{k-1} - nx^{n-1} > 0$ when $x \geq q > 1$, we obtain $w(x) > 0$ for all real $x > q$. Therefore, $d \leq q$, i.e., no root has magnitude strictly greater than q.

Now we prove by contradiction that $d \neq q$. Assume that $d = q$. Then, (28) becomes an equation, since $1 = d^k - d^n$ by (17). Thus,

$$|r_j^m - 1| = |r_j^m| - 1.$$

From the triangle inequality it follows that r_j^m is a real number. Therefore, we have $r_j^m = q^m$. Thus, for some integer $1 \leq s \leq m-1$, we have

$$r_j = q e^{2\pi s \sqrt{-1}/m}.$$

However, because m and n are relatively prime,

$$r_j^n = q^n e^{2\pi s n\sqrt{-1}/m} \neq q^n.$$

Therefore, $r_j^n(r_j^m - 1) \neq q^n(q^m - 1) = 1$, i.e., $w(r_j) \neq 0$. This is contradiction because r_j is a root of $w(x)$. □

In the following lemma, we calculate the coefficient c_1 for the dominant root $r_1 = q$ using the inverse of a Vandermonde matrix.

Lemma 5.11 *The coefficient c_1 in Lemma 5.9 is*

$$\left(\sum_{i=1}^{n} q^{-i} + \sum_{i=n+1}^{k} q^{k-i}\right)/(kq^{k-1} - nq^{n-1}).$$

Proof: We first give more notation. Let t and s be positive integers such that $1 \leq t, s \leq k$. We define $S_{t,s}$ as the sum of the products of t different roots not including r_s, that is,

$$S_{t,s} = \sum_{i_1<i_2<\ldots<i_t\in\{1,2,\ldots,k\}-\{s\}} r_{i_1} r_{i_2} \ldots r_{i_t}\,. \tag{29}$$

We define

$$S_{0,1} = 1\,, \tag{30}$$

and

$$S_{k,1} = 0\,. \tag{31}$$

We first give and solve the recurrence for $S_{t,1}$. We denote the coefficient of x^{t-1} in $w(x) = x^k - x^n - 1 = \prod_{i=1}^{k}(x - r_i)$ as $[\![x^{t-1}]\!]_{w(x)}$. Thus, we have the well known equation:

$$\sum_{i_1<i_2<\ldots<i_t\in\{1,2,\ldots,k\}} r_{i_1} r_{i_2} \ldots r_{i_t} = (-1)^t [\![x^{k-t}]\!]_{w(x)}. \tag{32}$$

Each product of roots in the summation in (32) either includes r_1 $(= q)$ or it does not, i.e.,

$$\sum_{i_1<i_2<\ldots<i_t\in\{1,2,\ldots,k\}} r_{i_1} r_{i_2} \ldots r_{i_t} = S_{t,1} + qS_{t-1,1}. \tag{33}$$

Thus, from (32) and (33) we obtain the recurrence

$$S_{t,1} + qS_{t-1,1} = (-1)^t [\![x^{k-t}]\!]_{w(x)}. \tag{34}$$

Because coefficients in $w(x)$ are 0 except for $[\![x^k]\!]_{w(x)} = 1$ and $[\![x^n]\!]_{w(x)} = [\![x^0]\!]_{w(x)} = -1$, we divide Recurrence (34) into two parts and solve each separately. Recall from (16) that $m = k - n$. The first part is when $t \in [1, m-1]$ and the second part is when $t \in [m, k-1]$. (Thus, when $t = m$, we need to confirm that the solution in the first part matches that in the second part.)

We solve the first part when $t \in [1, m-1]$. The base case is $t = 1$, that is,

$$S_{1,1} + qS_{0,1} = [\![x^{k-1}]\!]_{w(x)} = 0\,. \tag{35}$$

Observe that by (29) and (33), we have

$$\sum_{1\leq i\leq k} r_i = S_{1,1} + qS_{0,1} \quad \text{and} \quad \sum_{2\leq i\leq k} r_i = S_{1,1}. \tag{36}$$

Thus, from (36), we confirm that $S_{0,1} = 1$, and therefore from (35), we obtain

$$S_{1,1} = -q. \tag{37}$$

Because from (34),

$$S_{t,1} + qS_{t-1,1} = 0 \quad (1 \le t \le m-1)\,, \tag{38}$$

we also obtain, from (37) and (38),

$$S_{t,1} = (-q)^t. \tag{39}$$

We now solve the second part when $t \in [m, k-1]$. We start from k, that is,

$$S_{k,1} + qS_{k-1,1} = (-1)^k [\![x^0]\!]_{w(x)} = (-1)^{k-1}\,. \tag{40}$$

Observe that by (29) and (33), we have

$$r_1 r_2 \dots r_k = S_{k,1} + qS_{k-1,1} \quad \text{and} \quad r_2 r_3 \dots r_k = S_{k-1,1}. \tag{41}$$

From (41), we confirm that $S_{k,1} = 0$, and therefore from (40), we obtain $S_{k-1,1} = (-1)^{k-1}/q$. Because by (34),

$$S_{t+1,1} + qS_{t,1} = 0 \quad (m \le t \le k-1),$$

we obtain

$$S_{t,1} = (-1)^t q^{t-k}. \tag{42}$$

We now examine the special case where $t = m$ and $[\![x^n]\!]_{w(x)} = -1$, that is,

$$\begin{aligned} S_{m,1} + qS_{m-1,1} &= (-1)^m [\![x^n]\!]_{w(x)} \\ &= (-1)^{m+1}\,. \end{aligned} \tag{43}$$

We solved for all $S_{t,1}$ without using (43). We now confirm that our solution is consistent with (43). Notice that we get the solution in the first part, $S_{m-1,1} = (-q)^{m-1}$, and the solution in the second part, $S_{m,1} = (-1)^m q^{-n}$. In the following, we verify the solutions of $S_{m-1,1}$ and $S_{m,1}$ satisfy (43). Plugging

$$S_{m-1,1} = (-q)^{m-1} \quad \text{and} \quad S_{m,1} = (-1)^m q^{-n}$$

into (43), we obtain

$$\begin{aligned} S_{m,1} + qS_{m-1,1} &= (-1)^m q^{-n} + q(-q)^{m-1} \\ &= (-1)^m \frac{1 - q^{n+m}}{q^n} \end{aligned}$$

Because q is a root of $w(x)$, i.e.,

$$q^k = q^{m+n} = q^n + 1\,,$$

we confirm (43).

In summary, for all $1 \le t \le k-1$, we have

$$S_{t,1} = \begin{cases} (-q)^t, & \text{if } 1 \le t \le m-1; \\ (-1)^t q^{t-k}, & \text{if } m \le t \le k-1. \end{cases} \tag{44}$$

We now give even more notation. Define

$$g(x) = \prod_{i=2}^{k} (x - r_i). \tag{45}$$

We have $g(r_1) = g(q) = w'(q)$, because

$$w'(x) = \frac{d}{dx}\left[\prod_{i=1}^{k}(x - r_i)\right] = \sum_{j=1}^{k}\prod_{i=1, i\neq j}^{k}(x - r_i)$$

is a sum of k terms, but $k-1$ of these are 0 when $x = r_1 = q$. Thus, we obtain

$$(-1)^{k-1}g(r_1) = (-1)^{k-1}(kr_1^{k-1} - nr_1^{n-1}) = \prod_{i=2}^{k}(r_i - r_1). \tag{46}$$

Now we are ready to calculate the value of c_1. To do so, we define the Vandermonde matrix V:

$$V = \begin{pmatrix} 1 & r_1 & \cdots & r_1^{k-1} \\ 1 & r_2 & \cdots & r_2^{k-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & r_k & \cdots & r_k^{k-1} \end{pmatrix}.$$

Recall that (27) can be expressed as

$$(c_1, c_2, \ldots, c_k)V = (\alpha_0, \alpha_1, \ldots, \alpha_{k-1})\ .$$

Recall also that

$$\alpha_i = H(q^{i-k+1}) = 1 \quad (0 \leq i \leq k-1)$$

(because $q^{i-k+1} < 1$). Thus,

$$(c_1, \ldots, c_k) = (1, 1, \ldots, 1)\, V^{-1}, \tag{47}$$

i.e., c_1 can be calculated from the inverse matrix V^{-1}.

In order to calculate V^{-1}, we first present the well known result on how to calculate the determinant $|V|$ of Vandermonde matrix V.

$$|V| \;=\; \begin{vmatrix} 1 & r_1 & \cdots & r_1^{k-1} \\ 1 & r_2 & \cdots & r_2^{k-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & r_k & \cdots & r_k^{k-1} \end{vmatrix} = \prod_{1 \leq s < t \leq k}(r_t - r_s)\,. \tag{48}$$

We now give the inverse of V. Let $A_{i,j}$ be the submatrix of the transpose of the Vandermonde matrix V with the ith column and jth row removed, that is,

$$A_{i,j} = \begin{pmatrix} 1 & 1 & \cdots & 1 & 1 & \cdots & 1 \\ r_1 & r_2 & \cdots & r_{i-1} & r_{i+1} & \cdots & r_k \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ r_1^{j-1} & r_2^{j-1} & \cdots & r_{i-1}^{j-1} & r_{i+1}^{j-1} & \cdots & r_k^{j-1} \\ r_1^{j+1} & r_2^{j+1} & \cdots & r_{i-1}^{j+1} & r_{i+1}^{j+1} & \cdots & r_k^{j+1} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ r_1^{k-1} & r_2^{k-1} & \cdots & r_{i-1}^{k-1} & r_{i+1}^{k-1} & \cdots & r_k^{k-1} \end{pmatrix}.$$

Thus, V^{-1} can be represented by the determinants of $A_{i,j}$ and V, i.e.,

$$\begin{aligned} V^{-1} &= \frac{1}{|V|}\begin{pmatrix} (-1)^{1+1}|A_{1,1}| & \cdots & (-1)^{k+1}|A_{k,1}| \\ (-1)^{1+2}|A_{1,2}| & \cdots & (-1)^{k+2}|A_{k,2}| \\ \vdots & \ddots & \vdots \\ (-1)^{1+k}|A_{1,k}| & \cdots & (-1)^{k+k}|A_{k,k}| \end{pmatrix} \\ &= \left(\prod_{1\le s<t\le k}\frac{1}{r_t-r_s}\right)\left\{(-1)^{i+j}|A_{i,j}|\right\}_{i,j}. \end{aligned}$$

Thus, from (47), c_1 is the sum of the first column of inverse matrix V^{-1}, that is,

$$c_1 = \left(\prod_{1\le s<t\le k}\frac{1}{r_t-r_s}\right)\left(\sum_{j=1}^{k}(-1)^{1+j}|A_{1,j}|\right). \tag{49}$$

To calculate c_1, we first find $|A_{1,j}|$, which is given by the following claim:

Claim 5.12

$$|A_{1,j}| = S_{k-j,1}\prod_{2\le s<t\le k}(r_t-r_s).$$

Proof: When $j=1$,

$$|A_{1,1}| = \begin{vmatrix} r_2 & \cdots & r_k \\ \vdots & \ddots & \vdots \\ r_2^{k-1} & \cdots & r_k^{k-1} \end{vmatrix}.$$

By moving the common factors $r_2,\ldots,r_k$ out, we obtain

$$|A_{1,1}| = r_2\cdots r_k\begin{vmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ r_2^{k-2} & \cdots & r_k^{k-2} \end{vmatrix},$$

where the matrix is the transpose of Vandermonde matrix of size $k-1$. Thus, from (29) and (48), we obtain

$$|A_{1,1}| = r_2\cdots r_k\prod_{2\le s<t\le k}(r_t-r_s) = S_{k-1,1}\prod_{2\le s<t\le k}(r_t-r_s).$$

The case when $j\ge 2$ is more complicated than that $j=1$. In the following, we only consider $j=2$ because the other cases are analogous.

To solve $|A_{1,2}|$, we first perform matrix operations so that the first column becomes $\begin{pmatrix}1\\0\\\vdots\\0\end{pmatrix}$. Recall that

$$|A_{1,2}| = \begin{vmatrix} 1 & 1 & \cdots & 1 \\ r_2^2 & r_3^2 & \cdots & r_k^2 \\ \vdots & \vdots & \ddots & \vdots \\ r_2^{k-1} & r_3^{k-1} & \cdots & r_k^{k-1} \end{vmatrix}.$$

Beginning from the second row, we multiply each row by $-r_2$ and add it to the next row.

$$|A_{1,2}| = \begin{vmatrix} 1 & 1 & 1 & \cdots & 1 \\ r_2^2 & r_3^2 & r_4^2 & \cdots & r_k^2 \\ 0 & r_3^2(r_3-r_2) & r_4^2(r_4-r_2) & \cdots & r_k^2(r_k-r_2) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & r_3^{k-2}(r_3-r_2) & r_4^{k-2}(r_4-r_2) & \cdots & r_k^{k-2}(r_k-r_2) \end{vmatrix}.$$

For the second row, we multiply the first row by $-r_2^2$ and add it to the second row.

$$|A_{1,2}| = \begin{vmatrix} 1 & 1 & 1 & \cdots & 1 \\ 0 & r_3^2-r_2^2 & r_4^2-r_2^2 & \cdots & r_k^2-r_2^2 \\ 0 & r_3^2(r_3-r_2) & r_4^2(r_4-r_2) & \cdots & r_k^2(r_k-r_2) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & r_3^{k-2}(r_3-r_2) & r_4^{k-2}(r_4-r_2) & \cdots & r_k^{k-2}(r_k-r_2) \end{vmatrix}.$$

In this way, we reduce the dimension of $|A_{1,2}|$ to $k-2$, i.e.,

$$|A_{1,2}| = \begin{vmatrix} r_3^2-r_2^2 & r_4^2-r_2^2 & \cdots & r_k^2-r_2^2 \\ r_3^2(r_3-r_2) & r_4^2(r_4-r_2) & \cdots & r_k^2(r_k-r_2) \\ \vdots & \vdots & \ddots & \vdots \\ r_3^{k-2}(r_3-r_2) & r_4^{k-2}(r_4-r_2) & \cdots & r_k^{k-2}(r_k-r_2) \end{vmatrix}.$$

By moving out the common factors $r_3-r_2,\ldots,r_k-r_2$ in each column, we obtain:

$$|A_{1,2}| = \prod_{i=3}^{k}(r_i-r_2) \begin{vmatrix} r_3+r_2 & r_4+r_2 & \cdots & r_k+r_2 \\ r_3^2 & r_4^2 & \cdots & r_k^2 \\ \vdots & \vdots & \ddots & \vdots \\ r_3^{k-2} & r_4^{k-2} & \cdots & r_k^{k-2} \end{vmatrix}.$$

Now by splitting the first row, we obtain:

$$|A_{1,2}| = \prod_{i=3}^{k}(r_i-r_2) \begin{vmatrix} r_3 & r_4 & \cdots & r_k \\ r_3^2 & r_4^2 & \cdots & r_k^2 \\ \vdots & \vdots & \ddots & \vdots \\ r_3^{k-2} & r_3^{k-2} & \cdots & r_k^{k-2} \end{vmatrix} + \prod_{i=3}^{k}(r_i-r_2) \begin{vmatrix} r_2 & r_2 & \cdots & r_2 \\ r_3^2 & r_4^2 & \cdots & r_k^2 \\ \vdots & \vdots & \ddots & \vdots \\ r_3^{k-2} & r_4^{k-2} & \cdots & r_k^{k-2} \end{vmatrix}. \tag{50}$$

After moving out the common factors $r_3,\ldots,r_k$, the first term in (50) is a Vandermonde matrix of size $k-2$. For the second term in (50), we move out the common factor r_2 in the top row. Thus, using (48) we have

$$\begin{aligned} |A_{1,2}| &= \prod_{i=3}^{k}(r_i-r_2)r_3\cdots r_k \prod_{3\le s<t\le k}(r_t-r_s) + \prod_{i=3}^{k}(r_i-r_2)r_2 \begin{vmatrix} 1 & 1 & \cdots & 1 \\ r_3^2 & r_4^2 & \cdots & r_k^2 \\ \vdots & \vdots & \ddots & \vdots \\ r_3^{k-2} & r_4^{k-2} & \cdots & r_k^{k-2} \end{vmatrix} \\ &= r_3\cdots r_k \prod_{2\le s<t\le k}(r_t-r_s) + r_2\prod_{i=3}^{k}(r_i-r_2) \begin{vmatrix} 1 & 1 & \cdots & 1 \\ r_3^2 & r_4^2 & \cdots & r_k^2 \\ \vdots & \vdots & \ddots & \vdots \\ r_3^{k-2} & r_4^{k-2} & \cdots & r_k^{k-2} \end{vmatrix}. \end{aligned} \tag{51}$$

Notice that the determinant in (51) is a form of $A_{1,2}$ of size $k-2$. By the same method, we compute the determinant in (51) as

$$\begin{vmatrix} 1 & 1 & \cdots & 1 \\ r_3^2 & r_4^2 & \cdots & r_k^2 \\ \vdots & \vdots & \ddots & \vdots \\ r_3^{k-2} & r_4^{k-2} & \cdots & r_k^{k-2} \end{vmatrix} = r_4 \cdots r_k \prod_{3 \le s < t \le k} (r_t - r_s) + r_3 \prod_{i=4}^{k} (r_i - r_3) \begin{vmatrix} 1 & \cdots & 1 \\ r_4^2 & \cdots & r_k^2 \\ \vdots & \ddots & \vdots \\ r_4^{k-2} & \cdots & r_k^{k-2} \end{vmatrix}. \tag{52}$$

Thus, by plugging (52) into (51) we obtain

$$|A_{1,2}| = (r_3 \cdots r_k + r_2 r_4 \cdots r_k) \prod_{2 \le s < t \le k} (r_t - r_s) + r_2 r_3 \prod_{i=3}^{k} (r_i - r_2) \prod_{i=4}^{k} (r_i - r_3) \begin{vmatrix} 1 & \cdots & 1 \\ r_4^2 & \cdots & r_k^2 \\ \vdots & \ddots & \vdots \\ r_4^{k-2} & \cdots & r_k^{k-2} \end{vmatrix}.$$

With one more recursion, we obtain

$$\begin{aligned} |A_{1,2}| &= (r_3 \cdots r_k + r_2 r_4 \cdots r_k + r_2 r_3 r_5 \cdots r_k) \prod_{2 \le s < t \le k} (r_t - r_s) \\ &+ r_2 r_3 r_4 \prod_{i=3}^{k} (r_i - r_2) \prod_{i=4}^{k} (r_i - r_3) \prod_{i=5}^{k} (r_i - r_4) \begin{vmatrix} 1 & \cdots & 1 \\ r_5^2 & \cdots & r_k^2 \\ \vdots & \ddots & \vdots \\ r_5^{k-2} & \cdots & r_k^{k-2} \end{vmatrix}. \end{aligned}$$

Repeating recursive steps and recalling that $S_{k-2,1} = r_3 r_4 \cdots r_k + r_2 r_4 \cdots r_k + \cdots + r_2 r_3 \cdots r_{k-1}$ from the definition of $S_{t,s}$ in (29), we obtain

$$|A_{1,2}| = S_{k-2,1} \prod_{2 \le s < t \le k} (r_t - r_s).$$

We thus establish the claim. □

By combining Claim 5.12 and (49) we obtain

$$c_1 = \left(\prod_{1 \le i < j \le k} (r_j - r_i)^{-1} \right) \left(\sum_{i=1}^{k} (-1)^{i+1} S_{k-i,1} \prod_{2 \le s < t \le k} (r_t - r_s) \right). \tag{53}$$

Multiplying through and separating two cases of $S_{i,1}$ in (44), we obtain

$$c_1 = \prod_{2 \le j \le k} (r_j - r_1)^{-1} \left(\sum_{i=1}^{n} + \sum_{i=n+1}^{k} \right) (-1)^{i+1} S_{k-i,1}. \tag{54}$$

Plugging (44) into (54), we have

$$c_1 = \prod_{2 \le j \le k} (r_j - r_1)^{-1} \left[\sum_{i=1}^{n} (-1)^{k+1} q^{-i} + \sum_{i=n+1}^{k} (-1)^{k+1} q^{k-i} \right]. \tag{55}$$

Plugging (46) into (55), we solve for c_1:

$$c_1 = \left(\sum_{i=1}^{n} q^{-i} + \sum_{i=n+1}^{k} q^{k-i} \right) / (kq^{k-1} - nq^{n-1}), \tag{56}$$

which concludes the proof of Lemma 5.11. □

Thus, the value of c_1 is as claimed in the hypothesis of Lemma 5.5. After establishing the properties of $H(x)$, we can now give a proof of Lemma 5.5.

Proof of Lemma 5.5: To complete the proof we only need to show that

$$H(x) \leq (c_1 + \varepsilon)q^k x + O(1),$$

when $x \geq O(k/\varepsilon)$.

Observe that $H(x)$ is monotonically increasing and for each $x > 1$, we have $x \leq q^{\lceil \log_q x \rceil} \leq qx$. Thus, we bound $H(x)$ as follows:

$$H(x) \leq H(q^{\lceil \log_q x \rceil}) = \alpha_{\lceil \log_q x \rceil + k - 1}, \tag{57}$$

where the second equality is the definition of α_i. We denote $\lceil \log_q x \rceil + k - 1$ as i to simplify notation in the rest of the proof. Recall that $\alpha_i = \sum_{j=1}^{k} c_j r_j^i$ and that $r_1 = q$ is the dominant root. Thus, we have

$$\frac{\alpha_i}{q^i} = c_1 + \sum_{j=2}^{k} c_j \left(\frac{r_j}{q}\right)^i. \tag{58}$$

Because r_1 is the dominant root and the other roots have absolute value less than 1, we have

$$\sum_{j=2}^{k} c_j \left(\frac{r_j}{q}\right)^i \leq O\left(\frac{k}{q^i}\right).$$

Because $i = \lceil \log_q x \rceil + k - 1$, we have $q^i > x$. Thus, for any $\varepsilon > 0$, we can choose $x \geq O(k/\varepsilon)$ such that the last term in (58) is arbitrary small, that is,

$$\sum_{j=2}^{k} c_j \left(\frac{r_j}{q}\right)^i \leq O\left(\frac{k}{x}\right) \leq \varepsilon.$$

Therefore, we obtain $\alpha_i = (c_1 + \varepsilon)q^i$. Combining with (57), we have

$$H(x) \leq (c_1 + \varepsilon)q^{\lceil \log_q x \rceil + k - 1}. \tag{59}$$

Finally, plugging $q^{\lceil \log_q x \rceil} \leq qx$ into (59), we obtain, for $x \geq O(k/\varepsilon)$,

$$H(x) \leq (c_1 + \varepsilon)q^k x$$

as claimed. □

Bounding the Block-Boundary Crossing Function

We now give the memory-transfer cost from block-boundary crossings, and we show that it is dominated by the the memory-transfer cost from the path length. We consider the case when $a \geq 1/4$, which includes the best layouts. Using similar reasoning for computing the path-length cost, we obtain the following theorem:

Theorem 5.13 (Block-Boundary Crossing Cost) *The expected number of block-boundary-induced memory transfers $\mathcal{C}(x)$ on a search is at most $O(\lg \lg B / \lg B) \log_B x$ when $1/4 \leq a < 1/2$.*

Proof: The idea to bound $\mathcal{C}(x)$ is the same as that in bounding the path-length cost. That is, we solve the same Recurrence (25) except for the base case α_i $(0 \leq i \leq k-1)$, which from (3) is

$$\frac{2^{q^{i-k+1} \lg B} - 2}{B}$$

instead of 1.

Thus, we obtain the new value of coefficient c_1' which is similar to (53):

$$c_1' = \left(\prod_{1 \le i < j \le k} \frac{1}{r_j - r_i} \right) \left(\sum_{i=1}^{k} \frac{2^{q^{i-k} \lg B} - 2}{B} (-1)^{i+1} S_{k-i,1} \prod_{2 \le s < t \le k} (r_s - r_t) \right).$$

Multiplying through and separating the numerator, we have

$$c_1' = \prod_{2 \le j \le k} \frac{1}{r_j - r_1} \sum_{i=1}^{k} \frac{2^{q^{i-k} \lg B}}{B} (-1)^{i+1} S_{k-i,1} - \frac{2}{B} \prod_{2 \le j \le k} \frac{1}{r_j - r_1} \sum_{i=1}^{k} (-1)^{i+1} S_{k-i,1}. \tag{60}$$

Because the second term in (60) is $2c_1/B = O(1/B)$ by (53), we obtain

$$c_1' = \left(\prod_{2 \le j \le k} \frac{1}{r_j - r_1} \right) \left(\sum_{i=1}^{k} \frac{2^{q^{i-k} \lg B}}{B} (-1)^{i+1} S_{k-i,1} \right) - O\left(\frac{1}{B} \right). \tag{61}$$

In order to bound c_1', we count the number of terms in the summation in (61), i.e., the number of values of i, such that

$$\frac{2^{q^{i-k} \lg B}}{B} > \frac{1}{\lg B}.$$

That is, we determine the smallest value of i, such that

$$q^{i-k} > \frac{\lg(B/\lg B)}{\lg B} = 1 - \frac{\lg \lg B}{\lg B}.$$

Thus, we solve that

$$i - k > \ln\left(1 - \frac{\lg \lg B}{\lg B}\right) \frac{\lg e}{\lg q}. \tag{62}$$

We now estimate the previous expression. Recall that $\ln(1-x) > -x$ for $0 < x < 1$. Thus, from (62), we have

$$i - k > -\frac{\lg e \lg \lg B}{\lg q \lg B}.$$

If we denote

$$\nu = k - \frac{\lg e \lg \lg B}{\lg q \lg B}, \tag{63}$$

then we have

$$\frac{2^{q^{i-k} \lg B}}{B} \begin{cases} \le \dfrac{1}{\lg B}, & \text{when } 1 \le i \le \nu; \\ > \dfrac{1}{\lg B}, & \text{when } \nu < i \le k. \end{cases}$$

Separating the summation in (61) at ν, we obtain

$$c_1' \le \prod_{2 \le j \le k} \frac{1}{r_j - r_1} \sum_{i=1}^{\nu} \frac{1}{\lg B} (-1)^{i+1} S_{k-i,1} + \prod_{2 \le j \le k} \frac{1}{r_j - r_1} \sum_{i=\nu}^{k} \frac{2^{q^{i-k} \lg B}}{B} (-1)^{i+1} S_{k-i,1} - O\left(\frac{1}{B} \right). \tag{64}$$

Again, from (53), the first term in (64) is less than $c_1/\lg B = O(1/\lg B)$. Thus, we have

$$c_1' \le O\left(\frac{1}{\lg B} \right) + \prod_{2 \le j \le k} \frac{1}{r_j - r_1} \sum_{i=\nu}^{k} \frac{2^{q^{i-k} \lg B}}{B} (-1)^{i+1} S_{k-i,1}. \tag{65}$$

Observe that $2^{q^{i-k}\lg B}/B \le 1$ for $1 \le i \le k$. We separate into the two cases of $S_{i,1}$ in (65) as we do earlier in (54), to obtain

$$c_1' \le O\left(\frac{1}{\lg B}\right) + (-1)^{k+1} \prod_{2\le j\le k} \frac{1}{r_j - r_1} \left(\sum_{\nu<i\le n} q^{-i} + \sum_{i\ge n+1,\, i>\nu} q^{k-i} \right).$$

Because both q^{-i} and q^{k-i} are less than q^k, we obtain

$$c_1' \le O\left(\frac{1}{\lg B}\right) + (-1)^{k+1} \prod_{2\le j\le k} \frac{1}{r_j - r_1} \sum_{\nu<i\le k} q^k.$$

Plugging (46) and (63) into the above inequality, we have

$$c_1' \le O\left(\frac{1}{\lg B}\right) + \frac{q^k}{kq^{k-1} - nq^{n-1}} \frac{\lg e \lg\lg B}{\lg q \lg B}. \tag{66}$$

We now prove the second term in (66) is $O(\lg\lg B/\lg B)$. Recalling that $q^k = 1/a$ from (15), we have

$$\lg q = -\frac{\lg a}{k} \tag{67}$$

and

$$q^n = q^k - 1 = \frac{1}{a} - 1\,. \tag{68}$$

Taking logs in (68), we obtain

$$n = \frac{\lg(1/a - 1)}{\lg q}\,. \tag{69}$$

Plugging (67) into (69), we obtain

$$n = k\frac{\lg(1/a - 1)}{\lg(1/a)}. \tag{70}$$

Notice that the function $f(x) = \lg(x-1)/\lg x$ is increasing for $x > 1$ because $f'(x) > 0$ for $x > 1$. Therefore, by the assumption $a \ge 1/4$ and (70), we have

$$n \le k\frac{\lg 3}{\lg 4} < \frac{4k}{5}. \tag{71}$$

Thus, observing that $q^{k-1} > q^{n-1} > 1$ and the above (71), we obtain

$$kq^{k-1} - nq^{n-1} > kq^{k-1} - \frac{4k}{5}q^{k-1} > k/5. \tag{72}$$

Combining (15), (67), and (72), we have

$$\frac{q^k}{kq^{k-1} - nq^{n-1}} \frac{\lg e}{\lg q} \le -\frac{1}{a}\frac{5}{k}\frac{k\lg e}{\lg a} = \frac{-5\lg e}{a\lg a} \le 10\lg e.$$

Finally, from (66) we obtain

$$c_1' \le O\left(\frac{1}{\lg B}\right) + O\left(\frac{\lg\lg B}{\lg B}\right) = O\left(\frac{\lg\lg B}{\lg B}\right),$$

as claimed. □

Now we present the main Theorem, which we obtain by combining Theorem 5.7 and 5.13.

Theorem 5.14 (Generalized vEB Layout) *The expected cost of a search in the generalized vEB layout is at most* $[\lg e + o(1)]\log_B N + O(\lg\lg B/\lg B)\log_B N + O(1)$.

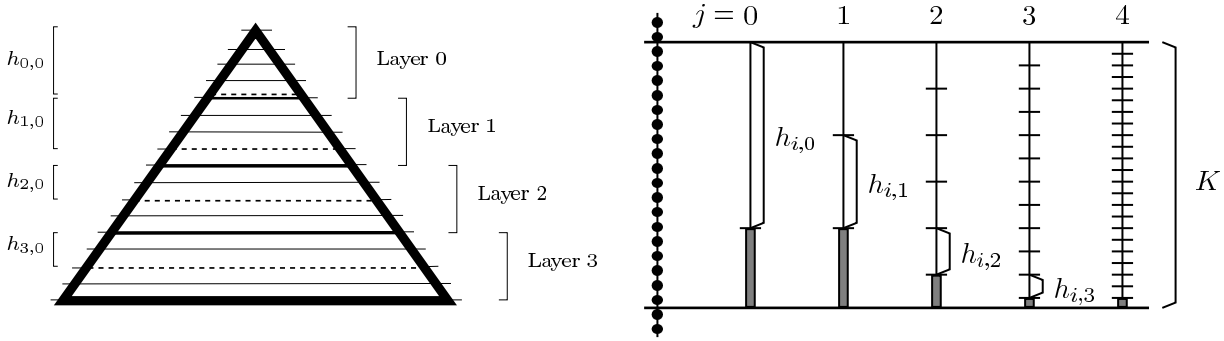

Figure 1: A multi-layer van Emde Boas tree decomposition. To the left is shown a tree decomposed into layers, and within each layer, the second recursive decomposition is indicated. Thinnest lines indicate tree levels; medium thin lines demarcate the first recursion layers; dashed lines indicate the height of the second recursive decomposition. To the right, the decomposition within a single layer i is shown for four additional recursive steps. The black circles on the left indicate tree levels, and the rest of the vertical lines each indicates a recursive level in the decomposition, with the recursive level increasing from left to right. The remainder part is shown in grey in each recursive level.

6 Multi-Layer van Emde Boas Layout

In this section we propose a somewhat more complex layout than in the previous part, but whose analysis is, in comparison, less cumbersome. First note that as observed in the proof of Theorem 3.1 if the size of the tree is a particular unfortunate multiple of the block size this can lead to a factor of two larger number of block transfers during a search as compared to the non-oblivious layout. Observe that as the block size B is unknown we cannot avoid this problem so long as we insist in decomposing a tree into equal-size fixed subtrees. Here we propose a recursive layout in which we select a large number of initial sizes for the second level of the decomposition.

Theorem 6.1 *For any constant $\varepsilon > 0$, there exists a cache-oblivious layout of a complete binary search tree with N leaves, such that any root-to-leaf traversal requires expected $O(1/\varepsilon)+(\lg e+\varepsilon+O(\lg\lg B/\lg B))\log_B N$ I/Os, where the expectation is taken over the starting position in memory of the layout.*

Proof: Since the tree is a complete binary tree, N is a power of two, and the height is $\lg N$. We group the nodes in the tree by depth into $H = 1 + \lg N$ levels numbered from zero to $H - 1$, with the root being at depth zero. The van Emde Boas layout splits the tree by depth at its mid-height level then recursively lays out of each the subtrees, which are again themselves split by depth into pieces of equal height. In contrast, in our layout we split the tree into L different layers of equal size and then within each layer aim for different heights of subtrees created during the recursion. For the second level in the decomposition, we split the layer in two, with the height of the top tree ranging across the layers from the entire layer down to half the layer. The further recursion within each layer is then defined by splitting into halves all subtrees at the current level of recursion, except the lowest subtrees. The latter, which we term the *remainder* subtrees, are given special consideration as described below. As an example, see Figure 1, in which the left part shows a tree decomposed into four equal sized layers, with each of these layers split into two parts at uneven depths. The right part shows the further recursive decomposition within a single layer.

We now give the full details. To simplify rounding issues in expressions involving levels and heights, we formulate our splitting strategy using intervals $[x, y) \subseteq [0, H)$, where $x, y \in \mathbb{R}$. We define the *levels spanned* by an interval $[x, y)$ as the levels numbered $\lfloor x \rfloor \ldots \lfloor y - 1 \rfloor$, and use this definition to transfer the interval-based decomposition of $[0, H)$ into a decomposition by depth of the H levels of the tree.

Let $\delta > 0$ be a constant given by $\delta = \lg(1 + \varepsilon/\lg e)$. We partition the H levels into $L = \lceil 1/\delta \rceil$ *layers* of consecutive levels of the tree. Layer $i \in \{0, 1, ..., L - 1\}$ starts at level $\lfloor iK \rfloor$ where $K = H/L$. In other words, layer i is the levels spanned by the interval $[iK, (i + 1)K)$.

The entire tree is laid out in memory starting at a random memory position. First the nodes in layer zero are laid out, followed by the nodes in layer one, and so on. The nodes in a layer i are laid out by independently laying out each subtree of the layer rooted at level $\lfloor iK \rfloor$ of the tree.

Each such subtree is laid out in a recursive way similar to the standard van Emde Boas layout, except that the goal is to make the heights of the subtrees of the jth recursion in layer i be

$$h_{i,j} = \frac{K}{2^{j+i/L}},$$

for $j = 0, 1, 2, \ldots$. For fixed i (i.e., within a layer), $h_{i,j}$ decreases by a factor of two at each recursion. For fixed j (i.e., for a given depth of recursion), $h_{i,j}$ decreases by a factor of $2^{1/L}$ when advancing to the next layer $i + 1$. Note that this happens in a cyclic fashion if one from the last layer $i = L - 1$ continues to the first layer $i = 0$, but at the next recursive depth: $h_{L-1,j}/2^{1/L} = h_{0,j+1}$. More precisely, the heights of the subtrees are defined by intervals (via the levels spanned by them), and our goal is to find intervals of length $h_{i,j}$.

If we for a given layer i and recursive depth j use the lowest (in the root-to-leaf sense in the global tree we are laying out) interval as a *remainder interval*, we can achieve a length of exactly $h_{i,j}$ for all the other intervals. Specifically, for any layer i, consider the first recursion depth $j = 0$. The interval $[iK, (i+1)K)$ of the layer, of length K, is divided into a first interval of length $h_{i,0}$, and a second interval of length $K - h_{i,0}$. Note that $K/2 < h_{i,0} \leq K$ for all i. The latter interval is the remainder interval at the first recursion depth. At each further recursive step within the layer, all intervals except the remainder interval are divided into two halves of equal length. The remainder interval, of length X, is divided if $X > h_{i,j}$, in which case it is divided into two intervals of lengths $h_{i,j}$ and $X - h_{i,j}$, respectively. The latter of these becomes the remainder interval at the next recursive depth. Note that $X/2 < h_{i,j} < X$, since $h_{i,j}$ decreases by a factor of two for each new recursive depth. If $X \leq h_{i,j}$, the remainder interval is not divided, and continues as the remainder interval at the next recursive depth.

The recursion stops at intervals spanning less than two levels. If the remainder interval for this reason is not recursed on, there will be no remainder interval on larger recursive depths within the layer. It is easy to see that when halving an interval spanning at least two levels, each of the halves spans at least one level. This implies that leaf intervals of the recursion span exactly one level (except for the single leaf being the deepest remainder interval, which may happen to span zero levels).

We now consider a root-to-leaf traversal in a tree with a layout as described above. Let $H_0 = \lg B - \lg \lg B$. We first count the number of subtrees traversed on the path, for subtrees of the recursive layout defined by intervals $[x, y)$, where $y - x \leq H_0$. For each layer we find the smallest j such that $h_{i,j} \leq H_0$. By the cyclic progression of $h_{i,j}$, the L identified values $h_{i,j}$ will form a set

$$\frac{K_0}{2^{0/L}}, \frac{K_0}{2^{1/L}}, \frac{K_0}{2^{2/L}}, \cdots, \frac{K_0}{2^{(L-1)/L}},$$

where $H_0/2^{1/L} < K_0 \leq H_0$. Each $h_{i,j}$ satisfies $\frac{1}{2}H_0 < h_{i,j} \leq H_0$. Note that the smallest value $h_{i,j}$ in the above list is not necessarily from layer zero.

The number of subtrees on a path with an interval spanning at most H_0 levels is:

$$\sum_{k=0}^{L-1} \left\lceil \frac{K}{K_0/2^{k/L}} \right\rceil \leq L + \frac{K}{K_0} \sum_{i=0}^{L-1} 2^{k/L} \leq L + \frac{K}{K_0} \cdot \frac{1}{2^{1/L} - 1}.$$

Observing that $2^x - 1 \geq x \ln 2$, and substituting above, we get

$$L + \frac{K}{K_0} \cdot \frac{1}{2^{1/L} - 1} \leq L + \frac{K}{K_0 \cdot (1/L) \ln 2} \leq L + \frac{K}{K_0} \cdot L \cdot \lg e \leq L + \frac{H/L}{H_0/2^{1/L}} \cdot L \cdot \lg e = L + \frac{H}{H_0} 2^{1/L} \cdot \lg e.$$

Since a recursively laid out subtree with $s \leq B$ nodes consists of a consecutive sequence of s memory cells, the probability that the subtree is split over two blocks is $(s - 1)/B$, when taken over all possible initial

placements of the tree in memory. An interval $[x, y)$ where $y - x \leq H_0$ can at most span $\lceil H_0 \rceil$ levels, i.e. the corresponding subtrees contain at most $2^{\lceil H_0 \rceil} - 1$ nodes. Accessing such a tree can cause two I/Os with probability at most $(2^{\lceil H_0 \rceil} - 2)/B$.

We can now bound the total expected number of I/Os by

$$\begin{aligned}
&\left(L + \frac{H}{H_0} \cdot 2^{1/L} \cdot \lg e\right) \cdot \left(1 + \frac{2^{\lceil H_0 \rceil} - 2}{B}\right) \\
&= \left(\lceil 1/\delta \rceil + \frac{1 + \lg N}{\lg B - \lg\lg B} \cdot 2^{1/\lceil 1/\delta \rceil} \cdot \lg e\right) \cdot \left(1 + \frac{2^{\lceil \lg B - \lg\lg B \rceil} - 2}{B}\right) \\
&\leq \left(\lceil 1/\delta \rceil + \frac{1 + \lg N}{\lg B - \lg\lg B} \cdot 2^{\delta} \cdot \lg e\right) \cdot \left(1 + \frac{2}{\lg B}\right) \\
&= O(1/\varepsilon) + \log_B N \cdot \frac{1}{1 - \lg\lg B / \lg B} \cdot 2^{\delta} \cdot \lg e \cdot \left(1 + \frac{2}{\lg B}\right) \qquad (73) \\
&\leq O(1/\varepsilon) + \log_B N \cdot (1 + 2\lg\lg B / \lg B) \cdot 2^{\delta} \cdot \lg e \cdot \left(1 + \frac{2}{\lg B}\right) \qquad (74) \\
&= O(1/\varepsilon) + \log_B N \cdot (2^{\delta} \cdot \lg e + O(\lg\lg B / \lg B)) \\
&= O(1/\varepsilon) + \log_B N \cdot (\lg e + \varepsilon + O(\lg\lg B / \lg B)) \,,
\end{aligned}$$

where (73) for the first term used $\lg(1 + x) = \Theta(x)$ for $x \to 0$, and (74) for the second term used $1/(1 - x) \leq 1 + 2x$ for $x \in [0, 1/2]$ and w.l.o.g. assumed $2\lg\lg B \leq \lg B$. □

7 Conclusion

This paper gives upper and lower bounds on the cost of cache-oblivious searching; our bounds are tight to within low-order terms. Specifically, the paper shows a lower bound of $\lg e \log_B N$ memory transfers and an upper bound of $[\lg e + \varepsilon + O(\lg\lg B / \lg B)] \log_B N + O(1)$ expected memory transfers in the cache-oblivious model. In contrast, searching uses only $\log_B N + 1$ memory transfers in the DAM model. Interestingly, this $\lg e$ multiplicative slowdown in the cache-oblivious model compared to the DAM model comes about because the DAM model has only two levels of memory rather then because the memory parameters are unknown in the cache-oblivious model.

References

[1] P. Agarwal, L. Arge, A. Danner, and B. Holland-Minkley. Cache-oblivious data structures for orthogonal range searching. In *Proc. 19th ACM Symp. on Comp. Geom. (SOCG)*, pages 237–245, 2003.

[2] A. Aggarwal, B. Alpern, A. K. Chandra, and M. Snir. A model for hierarchical memory. In *Proc. of the 19th Ann. ACM Symp. on Theory of Computing (STOC)*, pages 305–314, 1987.

[3] A. Aggarwal, A. K. Chandra, and M. Snir. Hierarchical memory with block transfer. In *Proc. of the 28th Annual IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 204–216, 1987.

[4] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988.

[5] B. Alpern, L. Carter, E. Feig, and T. Selker. The uniform memory hierarchy model of computation. *Algorithmica*, 12(2–3):72–109, 1994.

[6] S. Alstrup, M. A. Bender, E. D. Demaine, M. Farach-Colton, J. I. Munro, T. Rauhe, and M. Thorup. Efficient tree layout in a multilevel memory hierarchy, 2002. `http://www.arXiv.org/abs/cs.DS/0211010`.

[7] M. Andrews, M. A. Bender, and L. Zhang. New algorithms for the disk scheduling problem. In *Proc. of the 37th Ann. Symp. on Foundations of Computer Science (FOCS)*, pages 580–589, 1996.

[8] M. Andrews, M. A. Bender, and L. Zhang. New algorithms for the disk scheduling problem. *Algorithmica*, 32(2):277–301, 2002.

[9] R. D. Barve and J. S. Vitter. A theoretical framework for memory-adaptive algorithms. In *Proc. of the 40th Ann. Symp. on Foundations of Computer Science (FOCS)*, pages 273–284, 1999.

[10] Bayer, R. and McCreight. Organization and maintenance of large ordered indexes. *Acta Informatica*, 1:173–189, 1972.

[11] M. Bender, R. Cole, and R. Raman. Exponential structures for cache-oblivious algorithms. In *Proc. 29th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 2380 of *LNCS*, pages 195–207, 2002.

[12] M. Bender, E. Demaine, and M. Farach-Colton. Efficient tree layout in a multilevel memory hierarchy. In *Proc. 10th Annual European Symp. on Algorithms (ESA)*, volume 2461 of *LNCS*, pages 165–173, 2002.

[13] M. A. Bender, G. S. Brodal, R. Fagerberg, D. Ge, S. He, H. Hu, J. Iacono, and A. López-Ortiz. The cost of cache-oblivious searching. In *Proc. 44th Ann. Symp. on Foundations of Computer Science (FOCS)*, pages 271–280, 2003.

[14] M. A. Bender, E. D. Demaine, and M. Farach-Colton. Cache-oblivious B-trees. *SIAM Journal on Computing*, 35(2):341–358, 2005.

[15] M. A. Bender, Z. Duan, J. Iacono, and J. Wu. A locality-preserving cache-oblivious dynamic dictionary. *Journal of Algorithms*, 3(2):115–136, 2004.

[16] M. A. Bender, M. Farach-Colton, J. T. Fineman, Y. R. Fogel, B. C. Kuszmaul, and J. Nelson. Cache-oblivious streaming B-trees. In *SPAA*, pages 81–92, 2007.

[17] M. A. Bender, M. Farach-Colton, and B. Kuszmaul. Cache-oblivious string B-trees. In *Proceedings of the 25th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 233–242, 2006.

[18] M. A. Bender, J. T. Fineman, S. Gilbert, and B. C. Kuszmaul. Concurrent cache-oblivious B-trees. In *SPAA '05: Proceedings of the seventeenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 228–237. ACM, 2005.

[19] G. S. Brodal, E. D. Demaine, J. T. Fineman, J. Iacono, S. Langerman, and J. I. Munro. Cache-oblivious dynamic dictionaries with optimal update/query tradeoff. In *Proc. 21st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1448–1456, 2010.

[20] G. S. Brodal and R. Fagerberg. Cache oblivious distribution sweeping. In *Proc. 29th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 2380 of *LNCS*, pages 426–438, 2002.

[21] G. S. Brodal and R. Fagerberg. Funnel heap - a cache oblivious priority queue. In *Proc. 13th Ann. International Symp. on Algorithms and Computation (ISAAC)*, volume 2518 of *LNCS*, pages 219–228, 2002.

[22] G. S. Brodal, R. Fagerberg, and R. Jacob. Cache oblivious search trees via binary trees of small height. In *Proc. 13th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 39–48, 2002.

[23] Comer, D. The ubiquitous B-tree. *ACM Computing Surveys*, 11(2):121–137, 1979.

[24] E. D. Demaine. Cache-oblivious algorithms and data structures. Lecture Notes from the EEF Summer School on Massive Data Sets, 2002.

[25] G. Franceschini and R. Grossi. Optimal worst-case operations for implicit cache-oblivious search trees. In *Proc. 8th International Workshop on Algorithms and Data Structures (WADS)*, volume 2748 of *Lecture Notes in Computer Science*, pages 114–126, 2003.

[26] G. Franceschini and R. Grossi. Optimal implicit dictionaries over unbounded universes. *Theory Comput. Syst.*, 39(2):321–345, 2006.

[27] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *40th Ann. Symp. on Foundations of Computer Science (FOCS)*, pages 285–297, 1999.

[28] J.-W. Hong and H. T. Kung. I/O complexity: The red-blue pebble game. In *Proc. of the 13th Ann. ACM Symp. on Theory of Computation (STOC)*, pages 326–333, 1981.

[29] D. E. Knuth. *The Art of Computer Programming: Fundamental Algorithms*, volume 1. Addison-Wesley, 3rd edition, 1997.

[30] P. Kumar. Cache oblivious algorithms. In U. Meyer, P. Sanders, and J. Sibeyn, editors, *Algorithms for Memory Hierarchies, LNCS 2625*, pages 193–212, 2003.

[31] H. Prokop. Cache oblivious algorithms. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, June 1999.

[32] N. Rahman, R. Cole, and R. Raman. Optimised predecessor data structures for internal memory. In *Proc. 5th Int. Workshop on Algorithm Engineering (WAE)*, volume 2141, pages 67–78, 2001.

[33] C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. *IEEE Computer*, 27(3):17–29, 1994.

[34] J. E. Savage. Extending the Hong-Kung model to memory hierachies. In *Proc. of the 1st Ann. International Conference on Computing and Combinatorics*, volume 959 of *Lecture Notes in Computer Science*, pages 270–281, 1995.

[35] S. Sen, S. Chatterjee, and N. Dumir. Towards a theory of cache-efficient algorithms. *J. ACM*, 49(6):828–858, 2002.

[36] R. C. Singleton. An algorithm for computing the mixed radix fast fourier transform. *IEEE Transactions on Audio and Electroacoustics*, AU-17(2):93–103, 1969.

[37] P. van Emde Boas. Preserving order in a forest in less than logarithmic time. In *Proc. of the 16th Ann. Symp. on Foundations of Computer Science (FOCS)*, pages 75–84, 1975.

[38] P. van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Information Processing Letters*, 6(3):80–82, 1977.

[39] J. S. Vitter. External memory algorithms and data structures: dealing with massive data. *ACM Computing Surveys*, 33(2), 2001.

[40] J. S. Vitter and E. A. M. Shriver. Algorithms for parallel memory I: Two-level memories. *Algorithmica*, 12(2–3):110–147, 1994.

[41] J. S. Vitter and E. A. M. Shriver. Algorithms for parallel memory II: Hierarchical multilevel memories. *Algorithmica*, 12(2–3):148–169, 1994.