The Batched Predecessor Problem in External Memory

Michael A. Bender[1][2], Martín Farach-Colton[2][3], Mayank Goswami[4], Dzejla Medjedovic[5], Pablo Montes[1], and Meng-Tsung Tsai[3]

[1] Stony Brook University, Stony Brook NY 11794, USA
{bender,pmontes}@cs.stonybrook.edu
[2] Tokutek, Inc.
[3] Rutgers University, Piscataway NJ 08854, USA
{farach,mtsung.tsai}@cs.rutgers.edu
[4] Max-Planck Institute for Informatics, Saarbrücken 66123, Germany
gmayank@mpi-inf.mpg.de
[5] Sarajevo School of Science and Technology, Sarajevo 71000, Bosnia-Herzegovina
dzejla.medjedovic@ssst.edu.ba

Abstract. We give lower and upper bounds for the batched predecessor problem in external memory. We study tradeoffs between the I/O budget to preprocess a dictionary S versus the I/O requirement to find the predecessor in S of each element in a query set Q. For Q polynomially smaller than S, we give lower bounds in three external-memory models: the I/O comparison model, the I/O pointer-machine model, and the indexability model.

In the comparison I/O model, we show that the batched predecessor problem needs $\Omega(\log_B n)$ I/Os per query element ($n = |S|$) when the preprocessing is bounded by a polynomial. With exponential preprocessing, the problem can be solved faster, in $\Theta((\log_2 n)/B)$ per element. We give the tradeoff that quantifies the minimum preprocessing required for a given searching cost.

In the pointer-machine model, we show that with $\mathcal{O}(n^{4/3-\varepsilon})$ preprocessing for any constant $\varepsilon > 0$, the optimal algorithm cannot perform asymptotically faster than a B-tree. In the indexability model, we exhibit the tradeoff between the redundancy r and access overhead α of the optimal indexing scheme, showing that to report all query answers in $\alpha(x/B)$ I/Os, $\log r = \Omega((B/\alpha^2)\log(n/B))$.

Our lower bounds have matching or nearly matching upper bounds.

1 Introduction

A ***static dictionary*** is a data structure that represents a set $S = \{s_1, s_2, \ldots, s_n\}$ subject to the following operations:

PREPROCESS(S):	Prepare a data structure to answer queries.
SEARCH(q, S):	Return TRUE if $q \in S$ and FALSE otherwise.
PREDECESSOR(q, S):	Return $\max_{s_i \in S}\{s_i < q\}$.

The traditional static dictionary can be extended to support batched operations. Let $Q = \{q_1, \ldots, q_x\}$. Then, the ***batched predecessor*** problem can be defined as follows:

BATCHEDPRED(Q, S): Return $A = \{a_1, \ldots, a_x\}$, where $a_i = \text{PREDECESSOR}(q_i, S)$.

This research was supported in part by NSF grants CCF 1114809, CCF 1114930, CCF 1217708, IIS 1247726, IIS 1247750, and IIS 1251137.

In this paper we prove lower bounds on the batched predecessor problem in ***external memory*** [3], that is, when the dictionary is too large to fit into main memory. We study tradeoffs between the searching cost and the cost to preprocess the underlying set S. We present our results in three models: the comparison-based I/O model [3], the pointer-machine I/O model [18], and the indexability model [10, 11].

We focus on query size $x \leq n^c$, for constant $c < 1$. Thus, the query Q can be large, but is still much smaller than the underlying set S. This query size is interesting because, although there is abundant parallelism in the batched query, common approaches such as linear merges and buffering [4, 6, 7] are suboptimal.

Our results show that the batched predecessor problem in external memory cannot be solved asymptotically faster than $\Omega(\log_B n)$ I/Os per query element if the preprocessing is bounded by a polynomial; on the other hand, the problem *can* be solved asymptotically faster, in $\Theta((\log_2 n)/B)$ I/Os, if we impose no constraints on preprocessing. These bounds stand in marked contrast to single-predecessor queries, where one search costs $\Omega(\log_B n)$ even if preprocessing is unlimited.

We assume that S and Q are sorted. Without loss of generality, Q is sorted because Q's sort time is subsumed by the query time. Without loss of generality, S is sorted, as long as the preprocessing time is slightly superlinear. We consider sorted S throughout the paper. For notational convenience, we let $s_1 < s_2 < \cdots < s_n$ and $q_1 < q_2 < \cdots < q_x$, and therefore $a_1 \leq a_2 \leq \cdots \leq a_x$.

Given that S and Q are sorted, an alternative interpretation of this paper is as follows: *how can we optimally merge two sorted lists in external memory?* Specifically, what is the optimal algorithm for merging two sorted lists in external memory when one list is some polynomial factor smaller than the other?

Observe that the naïve linear-scan merging is suboptimal because it takes $\Theta(n/B)$ I/Os, which is greater than the $\mathcal{O}(n^c \log_B n)$ I/Os of a B-tree-based solution. Buffer trees [4, 6, 7] also take $\Theta(n/B)$ I/Os during a terminal flush phase. This paper shows that with polynomial preprocessing, performing independent searches for each element in Q is optimal, but it is possible to do better for higher preprocessing.

Single and batched predecessor problems in RAM. In the comparison model, a single predecessor can be found in $\Theta(\log n)$ time using binary search. The batched predecessor problem is solved in $\Theta(x \log(n/x) + x)$ by combining merging and binary search [13, 14]. The bounds for both problems remain tight for any preprocessing budget.

Pătraşcu and Thorup [15] give tight lower bounds for single predecessor queries in the cell-probe model. We are unaware of prior lower bounds for the batched predecessor problem in the pointer-machine and cell-probe models.

Although batching does not help algorithms that rely on comparisons, Karpinski and Nekrich [12] give an upper bound for this problem in the word-RAM model (bit operations are allowed), which achieves $\mathcal{O}(x)$ for all batches of size $x = \mathcal{O}(\sqrt{\log n})$ ($\mathcal{O}(1)$ per element amortized) with superpolynomial preprocessing.

Batched predecessor problem in external memory. Dittrich et al. [8] consider multisearch problems where queries are simultaneously processed and satisfied by navigating through large data structures on parallel computers. They give a lower bound of $\Omega(x \log_B(n/x) + x/B)$ under stronger assumptions: no duplicates of nodes are allowed, the ith query has to finish before the $(i+1)$st query starts, and $x < n^{1/(2+\varepsilon)}$, for a constant $\varepsilon > 0$.

Buffering is a standard technique for improving the performance of external-memory algorithms [4, 6, 7]. By buffering, partial work on a set of operations can share an I/O, thus reducing the per-operation I/O cost. Queries can similarly be buffered. In this paper, the number of queries, x, is much smaller than the size, n, of the data structure being queried. As a result, as the partial work on the queries progresses, the query paths can diverge within the larger search structure, eliminating the benefit of buffering.

Goodrich et al. [9] present a general method for performing x simultaneous external memory searches in $\mathcal{O}((n/B + x/B)\log_{M/B}(n/B))$ I/Os when x is large. When x is small, this technique achieves $\mathcal{O}(x \log_B(n/B))$ I/Os with a modified version of the parallel fractional cascading technique of Tamassia and Vitter [19].

Results

We first consider the **comparison-based I/O model** [3]. In this model, the problem cannot be solved faster than $\Omega(\log_B n)$ I/Os per element if preprocessing is polynomial. That is, batching queries is not faster than processing them one by one. With exponential preprocessing, the problem can be solved faster, in $\Theta((\log_2 n)/B)$ I/Os per element. We generalize to show a query-preprocessing tradeoff.

Next we study the **pointer-machine I/O model** [18], which is less restrictive than the comparison I/O model in main memory, but more restrictive in external memory.[6] We show that with preprocessing at most $\mathcal{O}(n^{4/3-\varepsilon})$ for constant $\varepsilon > 0$, the cost per element is again $\Omega(\log_B n)$.

Finally, we turn to the more general **indexability model** [10, 11]. This model is frequently used to describe reporting problems, and it focuses on bounding the number of disk blocks that contain the answers to the query subject to the space limit of the data structure; the searching cost is ignored. Here, the ***redundancy parameter*** $\boldsymbol{r}$ measures the number of times an element is stored in the data structure, and the ***access overhead parameter*** $\boldsymbol{\alpha}$ captures how far the reporting cost is from the optimal.

We show that to report all query answers in $\alpha(x/B)$ I/Os, $r = (n/B)^{\Omega(B/\alpha^2)}$. The lower bounds in this model also hold in the previous two models. This result shows that it is impossible to obtain $\mathcal{O}(1/B)$ per element unless the space used by the data structure is exponential, which corresponds to the situation in RAM, where exponential preprocessing is required to achieve $\mathcal{O}(1)$ amortized time per query element [12].

The rest of this section formally outlines our results.

Theorem 1 (Lower and upper bound, unrestricted preprocessing, I/O comparison model). *Let S be a set of size n and Q a set of size $x \le n^c$, $0 \le c < 1$. In the I/O comparison model, computing* BATCHEDPRED(Q, S) *requires*

$$\Omega\left(\frac{x}{B}\log\frac{n}{xB} + \frac{x}{B}\right)$$

[6] An algorithm can perform arbitrary computations in RAM, but a disk block can be accessed only via a pointer that has been seen at some point in past.

I/Os in the worst-case, no matter the preprocessing. There exists a comparison-based algorithm matching this bound.

Traditional information-theoretic techniques give tight sorting-like lower bounds for this problem in the RAM model. In external memory, the analogous approach yields a lower bound of $\Omega\left(\frac{x}{B}\log_{M/B}\frac{n}{x}+\frac{x}{B}\right)$. On the other hand, repeated finger searching in a B-tree yields an upper bound of $\mathcal{O}(x\log_B n)$. Theorem 1 shows that both bounds are weak, and that in external memory this problem has a complexity that is between sorting and searching.

We can interpret results in the comparison model through the amount of information that can be learned from each I/O. For searching, a block input reduces the choices for the target position of the element by a factor of B, thus learning $\log B$ *bits of information.* For sorting, a block input learns up to $\log\binom{M}{B}=\Theta(B\log(M/B))$ bits (obtained by counting the ways that an incoming block can intersperse with elements resident in main memory). Theorem 1 demonstrates that in the batched predecessor problem, the optimal, unbounded-preprocessing algorithm learns B bits per I/O, more than for searching but less than for sorting.

The following theorem captures the tradeoff between the searching and preprocessing: at one end of the spectrum lies a B-tree ($j=1$) with linear construction time and $\log_B n$ searching cost per element, and on the other end is the parallel binary search ($j=B$) with exponential preprocessing cost and $(\log_2 n)/B$ searching cost. This tradeoff shows that even to obtain a performance that is only twice as fast as that of a B-tree, quadratic preprocessing is necessary. To learn up to $j\log(B/j+1)$ bits per I/O, the algorithm needs to spend $n^{\Omega(j)}$ in preprocessing.

Theorem 2 (Search-preprocessing tradeoff, I/O comparison model). *Let S be a set of size n and Q a set of size $x\le n^c$, $0\le c<1$. In the I/O comparison model, computing* BATCHEDPRED(Q,S) *in $\mathcal{O}((x\log_{B/j+1}n)/j)$ I/Os requires that* PREPROCESSING(S) *use $n^{\Omega(j)}$ blocks of space and I/Os.*

In order to show results in the I/O pointer-machine model, we define a graph whose nodes are the blocks on disk of the data structure and whose edges are the pointers between blocks. Since a block has size B, it can contain at most B pointers, and thus the graph is fairly sparse. We show that any such sparse graph has a large set of nodes that are far apart. If the algorithm must visit those well-separated nodes, then it must perform many I/Os. The crux of the proof is that, as the preprocessing increases, the redundancy of the data structure increases, thus making it hard to pin down specific locations of the data structure that must be visited. We show that if the data structure is reasonable in size—in our case $\mathcal{O}(n^{4/3-\varepsilon})$—then we can still find a large, well dispersed set of nodes that must be visited, thus establishing the following lower bound:

Theorem 3 (Lower bound, I/O pointer-machine model). *Let S be a set of size n. In the I/O pointer-machine model, if* PREPROCESSING(S) *uses $\mathcal{O}(n^{4/3-\varepsilon})$ blocks of space and I/Os, for any constant $\varepsilon>0$, then there exists a constant c and a set Q of size n^c such that computing* BATCHEDPRED(Q,S) *requires $\Omega(x\log_B(n/x)+x/B)$ I/Os.*

We note that in this theorem, c is a function of ε in that, the smaller the preprocessing, the larger the set for which the lower bound can be established.

Finally, we consider the indexability model [10, 11], where we show:

Theorem 4 **($r - \alpha$ tradeoff, indexability model).** *In the indexability model, any indexing scheme for the batched predecessor problem with access overhead $\alpha \leq \sqrt{B}/4$ has redundancy r satisfying $\log r = \Omega\left(B \log(n/B)/\alpha^2\right)$.*

A crucial ingredient in our proof is a well-known result from extremal set theory due to Rödl [16]. Partly due to the techniques we use and partly due to the generality of this model, we do not get lower bounds for query time exceeding $Q/\sqrt{B}$, which was possible in the previous two models.

2 Batched Predecessor in the I/O Comparison Model

In this section we give the lower bound for when preprocessing is unrestricted. Then we study the tradeoff between preprocessing and the optimal number of I/Os.

2.1 Lower Bounds for Unrestricted Space/Preprocessing

We begin with the definition of a search interval.

Definition 5 **(*Search interval*).** *At step t of an execution, the search interval $S_i^t = [\ell_i^t, r_i^t]$ for an element q_i comprises those elements in S that are still potential values for a_i, given the information that the algorithm has learned so far. When there is no ambiguity, the superscript t is omitted.*

Proof of Theorem 1 (Lower Bound). Consider the following problem instance:

1. For all q_i, $|S_i| = n/x$. That is, all elements in Q have been given the first $\log x$ bits of information about where they belong in S.
2. For all i and j ($1 \leq i \neq j \leq x$), $S_i \cap S_j = \emptyset$. That is, search intervals are disjoint.

We do not charge the algorithm for transferring elements of Q between main memory and disk. This accounting scheme is equivalent to allowing all elements of Q to reside in main memory at all times while still having the entire memory free for other manipulations. Storing Q in main memory does not provide the algorithm with any additional information, since the sorted order of Q is already known.

Now we only consider I/Os of elements in S. Denote a block being input as $b = (b_1, \ldots, b_B)$. Observe that every b_i ($1 \leq i \leq B$) belongs to at most one S_j. The element b_i acts as a **pivot** and helps q_j learn at most one bit of information—by shrinking S_j to its left or its right half.

Since a single pivot gives at most one bit of information, the entire block b can supply at most B bits, during an entire execution of $\textsc{BatchedPred}(Q, S)$.

We require the algorithm to identify the final block in S where each q_i belongs. Thus, the total number of bits that the algorithm needs to learn to solve the problem is $\Omega(x \log(n/xB))$. Along with the scan bound to output the answer, the minimum number of block transfers required to solve the problem is $\Omega\left(\frac{x}{B} \log \frac{n}{xB} + \frac{x}{B}\right)$. □

We devise a matching algorithm (assuming $B \log n < M$), which has $\mathcal{O}(n^B)$ preprocessing cost. This algorithm has huge preprocessing costs but establishes that the lower bound from Theorem 1 is tight.

Proof of Theorem 1 (Upper Bound). The algorithm processes Q in batches of size B, one batch at a time. A single batch is processed by simultaneously performing binary search on all elements of the batch until they find their rank within S.

In the preprocessing phase, the algorithm produces all $\binom{n}{B}$ possible blocks. The algorithm also constructs a perfectly balanced binary search tree T on S. The former takes at most $B\binom{n}{B}$ I/Os, which is $\mathcal{O}(n^B)$, while the latter has a linear cost. The $\binom{n}{B}$ blocks are laid out in a lexicographical order in external memory, and it takes $B \log n$ bits to address the location of any block. $\square$

2.2 Preprocessing-Searching Tradeoffs

We give a lower bound on the space required by the batched predecessor problem when the budget for searching is limited. We prove Theorem 2 by proving Theorem 7.

Definition 6. *An I/O containing elements of S is a **j-parallelization I/O** if j distinct elements of Q acquire bits of information during this I/O.*

Theorem 7. *For $x \leq n^{1-\varepsilon}$ $(0 < \varepsilon \leq 1)$ and a constant $\gamma > 0$, any algorithm that solves* $\text{BATCHEDPRED}(Q, S)$ *in at most $(\gamma x \log n)/(j \log(B/j + 1)) + x/B$ I/Os requires at least $\left(\varepsilon j n^{\varepsilon/2}/2e\gamma B\right)^{\varepsilon j/2\gamma}$ I/Os for preprocessing in the worst case.*

Proof. The proof is by a deterministic adversary argument. In the beginning, the adversary partitions S into x equal-sized chunks $C_1, \ldots, C_x$, and places each query element into a separate chunk (i.e., $S_i = C_i$). Now each element knows $\log x \leq (1 - \varepsilon) \log n$ bits of information. Each element is additionally given half of the number of bits that remain to be learned. This leaves another $T \geq (\varepsilon x \log n)/2$ total bits yet to be discovered. As in the proof of Theorem 1, we do not charge for the inputs of elements in Q, thereby stipulating that all remaining bits to be learned are through the inputs of elements of S.

Lemma 8. *To learn T bits in at most $(\gamma x \log n)/(j \log(B/j + 1))$ I/Os, there must be at least one I/O in which the algorithm learns at least $(j \log(B/j + 1))/a$ bits, where $a = 2\gamma/\varepsilon$.*

If multiple I/Os learn at least $(j \log(B/j + 1))/a$ bits, consider the last such I/O during the algorithm execution. Denote the contents of the I/O as $b_i = (p_1, \ldots, p_B)$.

Lemma 9. *The maximum number of bits an I/O can learn while parallelizing d elements is $d \log(B/d + 1)$.*

Lemma 10. *The I/O b_i parallelizes at least j/a elements.*

Proof. Given that the most bits an I/O can learn while parallelizing $j/a - 1$ elements is $(j/a - 1) \log (B/(j/a - 1) + 1)$ bits. For all $a \geq 1$ and $j \geq 2$, $\frac{j}{a} \log\left(\frac{B}{j} + 1\right) > \left(\frac{j}{a} - 1\right) \log\left(\frac{B}{j/a-1} + 1\right)$. Thus, we can conclude that with the block transfer of b_i, the algorithm must have parallelized strictly more than $j/a - 1$ distinct elements. $\square$

We focus our attention on an arbitrarily chosen group of j/a elements parallelized during the transfer of $b_i = \{p_1, \dots, p_B\}$, which we call $q_1, \dots, q_{j/a}$.

Lemma 11. *For every q_u parallelized during the transfer of b_i there is at least one pivot p_v, $1 \leq v \leq B$, such that $p_v \in S_u$.*

Consider the vector $V = (S_1, S_2, \dots, S_{j/a})$ where S_u denotes the search interval of q_u right before the input of b_i.

Each element of Q has acquired at least $(1 - \varepsilon/2) \log n$ bits, $(\varepsilon \log n)/2$ of which were given for free after the initial $(1 - \varepsilon) \log n$. For any i, the total number of distinct choices for S_i in the vector V is at least $n^{\varepsilon/2}$, because the element could have been sent to any of these $n^{\varepsilon/2}$-sized ranges in the initial n^{ε} range. We obtain the following:

Lemma 12. *The number of distinct choices for V at the time of parallelization is at least $n^{j\varepsilon/2a}$.*

Lemma 13. *For each of the $n^{j\varepsilon/2a}$ choices of $V = (S_1, \dots, S_{j/a})$ (arising from the $n^{\varepsilon/2}$ choices for each S_i), there must exist a block with pivots $p_1, p_2, \dots, p_{j/a}$, such that $p_k \in S_k$.*

If the algorithm did not preprocess a block for each vector choice, the adversary could scan all blocks, find a vector for which no block exists, and assign those search intervals to $q_1, \dots, q_{j/a}$, thus avoiding parallelization.

The same block can serve multiple vector choices, because the block has B elements and we are parallelizing only j/a elements. The next lemma quantifies the maximum number of vectors covered by one block.

Lemma 14. *A block can cover at most $\binom{B}{j/a}$ distinct vector choices.*

As a consequence, the minimum number of blocks the algorithm needs to preprocess is at least $n^{j\varepsilon/2a}/\binom{B}{j/a} \geq \left(n^{\varepsilon/2}/(eaB/j)\right)^{j/a}$. Substituting for the value of a, we get that the minimum preprocessing is at least $\left(\varepsilon j n^{\varepsilon/2}/2e\gamma B\right)^{\varepsilon j/2\gamma}$. □

Algorithms. An algorithm that runs in $\mathcal{O}((x \log n)/j \log(B/j + 1) + x/B)$ I/Os follows an idea similar to the optimal algorithm for unrestricted preprocessing. The difference is that we preprocess $\binom{n}{j}$ blocks, where each block correspond to a distinct combination of some j elements. The block will contain B/j evenly spaced pivots for each element. The searching algorithm uses batches of size j.

3 Batched Predecessor in the I/O Pointer-Machine Model

Here we analyze the batched predecessor problem in the I/O pointer-machine model. We show that if the preprocessing time is $\mathcal{O}(n^{4/3-\varepsilon})$ for any constant $\varepsilon > 0$, then there exists a query set Q of size x such that reporting BATCHEDPRED(Q, S) requires $\Omega(x/B + x \log_B n/x)$ I/Os. Before proving our theorem, we briefly describe the model.

I/O pointer machine model. The I/O pointer machine model [18] is a generalization of the pointer machine model introduced by Tarjan [21]. Many results in range reporting have been obtained in this model [1, 2].

To answer $\text{BATCHEDPRED}(Q, S)$, an algorithm preprocesses S and builds a data structure comprised of n^k blocks, where k is a constant to be determined later. We use a directed graph $\mathcal{G} = (V, E)$ to represent the n^k blocks and their associated directed pointers. Every algorithm that answers $\text{BATCHEDPRED}(Q, S)$ begins at the start node v_0 in V and at each step picks a directed edge to follow from those seen so far. Thus, the nodes in a computation are all reachable from v_0. Furthermore, each fetched node contains elements from S, and the computation cannot terminate until the visited set of elements is a superset of the answer set A. A node in V contains at most B elements from S and at most B pointers to other nodes.

Let $\mathcal{L}(W)$ be the union of the elements contained in a node set W, and let $\mathcal{N}(a)$ be the set of nodes containing element a. We say that a node set W ***covers*** a set of elements A if $A \subseteq \mathcal{L}(W)$. An algorithm for computing A can be modeled as the union of a set of paths from v_0 to each node in a node set W that covers A.

To prove a lower bound on $\text{BATCHEDPRED}(Q, S)$, we show that there is a query set Q whose answer set A requires many I/Os. In other words, for every node set W that covers A, a connected subgraph spanning W contains many nodes. We achieve this result by showing that there is a set A such that, for every pair of nodes $a_1, a_2 \in A$, the distance between $\mathcal{N}(a_1)$ and $\mathcal{N}(a_2)$ is large, that is, all the nodes in $\mathcal{N}(a_1)$ are far from all the nodes in $\mathcal{N}(a_2)$. Since the elements of A can appear in more than one node, we need to guarantee that the node set V of $\mathcal{G}$ is not too large; otherwise the distance between $\mathcal{N}(a_1)$ and $\mathcal{N}(a_2)$ can be very small. For example, if $|V| \geq \binom{n}{2}$, every pair of elements can share a node, and a data structure exists whose minimum pairwise distance between any $\mathcal{N}(a_1)$ and $\mathcal{N}(a_2)$ is 0.

First, we introduce two measures of distance between nodes in any (undirected or directed) graph $G = (V, E)$. Let $d_G(u, v)$ be the length of the shortest (di-)path from node u to node v in G. Furthermore, let $\Lambda_G(u, v) = \min_{w \in V} (d_G(w, u) + d_G(w, v))$. Thus, $\Lambda_G(u, v) = d_G(u, v)$ for undirected graphs, but not necessarily for directed graphs.

For each $W \subseteq V$, define $f_G(W)$ to be the minimum number of nodes in any connected subgraph H such that (1) the node set of H contains $W \cup \{v_0\}$ and (2) H contains a path from v_0 to each $v \in W$. Observe that $f_G(\{u, v\}) \geq \Lambda_G(u, v)$. The following lemma gives a more general lower bound for $f_G(W)$. In other words, the size of the graph containing nodes of W is linear in the minimum pairwise distance within W.

Lemma 15. *For any directed graph $G = (V, E)$ and any $W \subseteq V$ of size $|W| \geq 2$, $f_G(W) \geq r_W |W|/2$, where $r_W = \min_{u,v \in W, u \neq v} \Lambda_G(u, v)$.*

Proof Sketch. Consider the undirected version of G, and consider a TSP of the nodes in W. It must have length $r_W |W|$. Any tree that spans W must therefore have size at least $r_W |W|/2$. Finally, $f_G(W)$ contains a tree that spans W. □

Our next goal is to find a query set Q such that every node set W that covers the corresponded answer set A has a large r_W. The answer set A will be an independent set of a certain kind, that we define next. For a directed graph $G = (V, E)$ and an integer $r > 0$, we say that a set of nodes $I \subseteq V$ is ***r-independent*** if $\Lambda_G(u, v) > r$ for all $u, v \in I$ where $u \neq v$. The next lemma guarantees a substantial r-independent set.

Lemma 16. *Given a directed graph $G = (V, E)$, where each node has out-degree at most $B \geq 2$, there exists an r-independent set I of size at least $\frac{|V|^2}{|V|+4r|V|B^r}$.*

Proof. Construct an undirected graph $H = (U, F)$ such that $U = V$ and $(u, v) \in F$ iff $\Lambda_G(u, v) \in [1, r]$. Then, H has at most $2r|V|B^r$ edges. By Turán's Theorem [20], there exists an independent set of the desired size in H, which corresponds to an r-independent set in G, completing the proof. $\square$

In addition to r-independence, we want the elements in A to occur in few blocks, in order to control the possible choices of the node set W that covers A. We define the ***redundancy*** of an element a to be $|\mathcal{N}(a)|$. Because there are n^k blocks and each block has at most B elements, the average redundancy is $\mathcal{O}(n^{k-1}B)$. We say that an element has ***low redundancy*** if its redundancy is at most twice the average. We show that there exists an r-independent set I of size n^{ε} (here ε depends on r) such that no two blocks share the same low-redundancy element. We will then construct our query set Q using this set of low-redundancy elements in this r-independent set.[7]

Finally, we add enough edges to place all occurrences of every low-redundancy element within $\rho < r/2$ of all other occurrences of that element. We show that we can do this by adding few edges to each node, therefore maintaining the sparsity of G. Since this augmented graph also contains a large r-independent set, all the nodes of this set cannot share any low-redundancy elements.

The following lemma shows that nodes sharing low-redundancy elements can be connected with low diameter and small degrees.

Lemma 17. *For any $k > 0$ and $m > k$ there exists an undirected k-regular graph H of order m having diameter $\log_{k-1} m + o(\log_{k-1} m)$.*

Proof. In [5], Bollobás shows that a random k-regular graph has the desired diameter with probability close to 1. Thus there exists some graph satisfying the constraints. $\square$

Consider two blocks B_1 and B_2 in the r-independent set I above, and let a and b be two low-redundancy elements such that $a \in B_1, b \notin B_1$ and $a \notin B_2, b \in B_2$. Any other pair of blocks $B_1^{'}$ and $B_2^{'}$ that contain a and b respectively must be at least $(r - 2\rho)$ apart, since $B_i^{'}$ is at most ρ apart from B_i. By this argument, every node set W that covers A has $r_W \geq (r - 2\rho)$. Now, by Lemma 15, we get a lower bound of $\Omega((r - 2\rho)|W|)$ on the query complexity of Q. We choose $r = c_1 \log_B(n/x)$ and get $\rho = c_2 \log_B(n/x)$ for appropriate constants $c_1 > 2c_2$. This is the part where we require the assumption that $k < 4/3$ as shown in Theorem 3, where n^k was the size of the entire data structure. We then apply Lemma 16 to obtain that $|W| = \Omega(x)$.

Proof of Theorem 3. We partition S into S_ℓ and S_h by the redundancy of elements in these n^k blocks and claim that there exists $A \subseteq S_\ell$ such that query time for the corresponded Q matches the lower bound.

Let S_ℓ be the set of elements of redundancy no more than $2Bn^k/n$ (i.e., twice of the average redundancy). The rest of elements belong to S_h. By the Markov inequality, we

[7] Our construction does not work if the query set contains high redundancy elements, because high redundancy elements might be placed in every block.

have $|S_\ell| = \Theta(n)$ and $|S_h| \le n/2$. Let $\mathcal{G} = (V, E)$ represent the connections between the n^k blocks as the above stated. We partition V into V_1 and V_2 such that V_1 is the set of blocks containing some elements in S_ℓ and $V_2 = V \setminus V_1$. Since each block can at most contain B elements in S_ℓ, $|V_1| = \Omega(n/B)$.

Then, we add some additional pointers to $\mathcal{G}$ and obtain a new graph $\mathcal{G}'$ such that, for each $e \in S_\ell$, every pair $u, v \in \mathcal{N}(e)$ has small $\Lambda_{\mathcal{G}'}(u, v)$. We achieve this by, for each $e \in S_\ell$, introducing graph H_e to connect all the n^k blocks containing element e such that the diameter in H_e is small and the degree for each node in H_e is $\mathcal{O}(B^\delta)$ for some constant δ. By Lemma 17, the diameter of H_e can be as small as

$$\rho \le \frac{1}{\delta} \log_B |H_e| + o(\log_B |H_e|) \le \frac{k-1}{\delta} \log_B n + o(\log_B n).$$

We claim that the graph $\mathcal{G}'$ has a $(2\rho + \varepsilon)$-independent set of size n^c, for some constants $\varepsilon, c > 0$. For the purpose, we construct an undirected graph $H(V_1, F)$ such that $(u, v) \in F$ iff $\Lambda_{\mathcal{G}'}(u, v) \le r$. Since the degree of each node in $\mathcal{G}'$ is bounded by $\mathcal{O}(B^{\delta+1})$, by Lemma 16, there exists an r-independent set I of size

$$|I| \ge \frac{|V_1|^2}{|V_1| + 4r|V|\mathcal{O}(B^{r(\delta+1)})} \ge \frac{n^{2-k}}{4r\mathcal{O}(B^{r(\delta+1)+2})} = n^c.$$

Then, $r = ((2-k-c)\log_B n)/(\delta+1) + o(\log_B n)$. To satisfy the condition made in the claim, let $r > 2\rho$. Hence, $(2-k-c)/(\delta+1) > 2(k-1)/\delta$. Then, $k \to 4/3$ for sufficiently large δ. Observe that, for each $e \in S_\ell$, e is contained in at most one node in I; in addition, for every pair $e_1, e_2 \in S_\ell$ where e_1, e_2 are contained in separated nodes in I, then $\Lambda_{\mathcal{G}'}(u, v) \ge \varepsilon$ for any $u \ni e_1, v \ni e_2$. By Lemma 15, we are done. $\square$

4 Batched Predecessor in the Indexability Model

This section analyzes the batched predecessor problem in the indexability model [10, 11]. This model is used to analyze reporting problems by focusing on the number of blocks that an algorithm must access to report all the query results. Lower bounds on queries are obtained solely based on how many blocks were preprocessed. The search cost is ignored—the blocks containing the answers are given to the algorithm for free.

A ***workload*** is given by a pair $\mathcal{W} = (S, \mathcal{A})$, where S is the set of n input objects, and $\mathcal{A}$ is a set of subsets of S—the output to the queries. An ***indexing scheme*** $\mathcal{I}$ for a given workload $\mathcal{W}$ is given by a collection $\mathcal{B}$ of B-sized subsets of S such that $S = \cup\mathcal{B}$; each $b \in \mathcal{B}$ is called a block.

An indexing scheme has two parameters associated with it. The first parameter, called the ***redundancy***, represents the average number of times an element is replicated (i.e., an indexing scheme with redundancy r uses $r\lceil n/B\rceil$ blocks). The second parameter is called the ***access overhead***. Given a query with answer A, the query time is $\min\{|\mathcal{B}'| : \mathcal{B}' \subseteq \mathcal{B}, A \subseteq \cup\mathcal{B}'\}$, because this is the minimum number of blocks that contain all the answers to the query. If the size of A is x, then the best indexing scheme would require a query time of $\lceil x/B\rceil$. The access overhead of an indexing scheme is the factor by which it is suboptimal. An indexing scheme with access overhead α uses $\alpha\lceil x/B\rceil$ I/Os to answer a query of size x in the worst case.

Every lower bound in this model applies to our previous two models as well. To show the tradeoff between α and r, we use the Redundancy Theorem from [11, 17]:

Theorem 18 (Redundancy Theorem [11, 17]). *For a workload $\mathcal{W} = (S, \mathcal{A})$ where $\mathcal{A} = \{A_1, \cdots, A_m\}$, let $\mathcal{I}$ be an indexing scheme with access overhead $\alpha \leq \sqrt{B}/4$ such that for any $1 \leq i, j \leq m$, $i \neq j$, $|A_i| \geq B/2$ and $|A_i \cap A_j| \leq B/(16\alpha^2)$. Then the redundancy of $\mathcal{I}$ is bounded by $r \geq \frac{1}{12n}\sum_{i=1}^{m}|A_i|$.*

Proof of Theorem 4. For the sake of the lower bound, we restrict to queries where all the reported predecessors reported are distinct. To use the redundancy theorem, we want to create as many queries as possible.

Call a family of k-element subsets of S β-sparse if any two members of the family intersect in less than β elements. The size $C(n, k, \beta)$ of a maximal β-sparse family is crucial to our analysis. For a fixed k and β this was conjectured to be asymptotically equal to $\binom{n}{\beta}/\binom{k}{\beta}$ by Erdös and Hanani and later proven by Rödl in [16]. Thus, for large enough n, $C(n, k, \beta) = \Omega(\binom{n}{\beta}/\binom{k}{\beta})$.

We now pick a $(B/2)$-element, $B/(16\alpha^2)$-sparse family of S, where α is the access overhead of $\mathcal{I}$. The result in [16] gives us that

$$C\left(n, \frac{B}{2}, \frac{B}{16\alpha^2}\right) = \Omega\left(\binom{n}{B/(16\alpha^2)}\Big/\binom{B/2}{B/(16\alpha^2)}\right).$$

Thus, there are at least $(2n/eB)^{B/(16\alpha^2)}$ subsets of size $B/2$ such that any pair intersects in at most $B/(16\alpha^2)$ elements. The Redundancy Theorem then implies that the redundancy r is greater than or equal to $(n/B)^{\Omega(B/\alpha^2)}$, completing the proof. $\square$

We describe an indexing scheme that is off from the lower bound by a factor α.

Theorem 19 (Indexing scheme for the batched predecessor problem). *Given any $\alpha \leq \sqrt{B}$, there exists an indexing scheme $\mathcal{I}_\alpha$ for the batched predecessor problem with access overhead α^2 and redundancy $r = \mathcal{O}((n/B)^{B/\alpha^2})$*

Proof. Call a family of k-element subsets of S β-dense if any subset of S of size β is contained in at least one member from this family. Let $c(n, k, \beta)$ denote the minimum number of elements of such a β-dense family. Rödl [16] proves that for a fixed k and β,

$$\lim_{n\to\infty} c(n, k, \beta)\binom{k}{\beta}\binom{n}{\beta}^{-1} = 1,$$

and thus, for large enough n, $c(n, k, \beta) = \mathcal{O}(\binom{n}{\beta}/\binom{k}{\beta})$.

The indexing scheme $\mathcal{I}_\alpha$ consists of all sets in a B-element, (B/α^2)-dense family. By the above, the size of $\mathcal{I}_\alpha$ is $\mathcal{O}((n/B)^{B/\alpha^2})$.

Given a query answer $A = \{a_1, \cdots, a_x\}$ of size x, fix $1 \leq i < \lceil x/B \rceil$ and consider the B-element sets $C_i = \{a_{(i-1)B}, \cdots, a_{iB}\}$ ($C_{\lceil x/B \rceil}$ may have less than B elements). Since $\mathcal{I}_\alpha$ is an indexing scheme, we are told all the blocks in $\mathcal{I}_\alpha$ that contain the a_is. By construction, there exists a block in $\mathcal{I}_\alpha$ that contains a $1/\alpha^2$ fraction of C_i. In at most α^2 I/Os we can output C_i, by reporting B/α^2 elements in every I/O. The number of I/Os needed to answer the entire answer A is thus $\alpha^2\lceil x/B \rceil$, which proves the theorem. $\square$

References

1. Afshani, P., Arge, L., Larsen, K.D.: Orthogonal range reporting: Query lower bounds, optimal structures in 3-d, and higher-dimensional improvements. In: 26th Annual Symposium on Computational Geometry (SoCG). pp. 240–246 (2010)
2. Afshani, P., Arge, L., Larsen, K.G.: Higher-dimensional orthogonal range reporting and rectangle stabbing in the pointer machine model. In: 28th Annual Symposium on Computational Geometry (SoCG). pp. 323–332 (2012)
3. Aggarwal, A., Vitter, Jeffrey, S.: The input/output complexity of sorting and related problems. Commun. ACM 31, 1116–1127 (1988)
4. Arge, L.: The buffer tree: A technique for designing batched external data structures. Algorithmica 37(1), 1–24 (2003)
5. Bollobás, B., Fernandez de la Vega, W.: The diameter of random regular graphs. Combinatorica 2(2), 125–134 (1982)
6. Brodal, G.S., Fagerberg, R.: Lower bounds for external memory dictionaries. In: 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). pp. 546–554 (2003)
7. Buchsbaum, A.L., Goldwasser, M., Venkatasubramanian, S., Westbrook, J.R.: On external memory graph traversal. In: 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). pp. 859–860 (2000)
8. Dittrich, W., Hutchinson, D., Maheshwari, A.: Blocking in parallel multisearch problems (extended abstract). In: 10th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA). pp. 98–107 (1998)
9. Goodrich, M.T., Tsay, J.J., Cheng, N.C., Vitter, J., Vengroff, D.E., Vitter, J.S.: External-memory computational geometry. In: 1993 IEEE 34th Annual Foundations of Computer Science (FOCS). pp. 714–723 (1993)
10. Hellerstein, J.M., Koutsoupias, E., Papadimitriou, C.H.: On the analysis of indexing schemes. In: 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS). pp. 249–256 (1997)
11. Hellerstein, J.M., Koutsoupias, E., Miranker, D.P., Papadimitriou, C.H., Samoladas, V.: On a model of indexability and its bounds for range queries. J. ACM 49, 35–55 (2002)
12. Karpinski, M., Nekrich, Y.: Predecessor queries in constant time? In: 13th Annual European Conference on Algorithms (ESA). pp. 238–248 (2005)
13. Knudsen, M., Larsen, K.: I/O-complexity of comparison and permutation problems. Master's thesis, DAIMI (November 1992)
14. Knuth, D.E.: The Art of Computer Programming: Sorting and Searching, vol. 3. Addison Wesley (1973)
15. Pătraşcu, M., Thorup, M.: Time-space trade-offs for predecessor search. In: 38th Annual ACM Symposium on Theory of Computing (STOC). pp. 232–240 (2006)
16. Rödl, V.: On a packing and covering problem. European Journal of Combinatorics 6(1), 69–78 (1985)
17. Samoladas, V., Miranker, D.P.: A lower bound theorem for indexing schemes and its application to multidimensional range queries. In: 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS). pp. 44–51 (1998)
18. Subramanian, S., Ramaswamy, S.: The p-range tree: A new data structure for range searching in secondary memory. In: Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). pp. 378–387 (1995)
19. Tamassia, R., Vitter, J.S.: Optimal cooperative search in fractional cascaded data structures. In: Algorithmica. pp. 307–316 (1990)
20. Tao, T., Vu, V.H.: Additive Combinatorics. Cambridge University Press (2009)
21. Tarjan, R.E.: A class of algorithms which require nonlinear time to maintain disjoint sets. Journal of Computer and System Sciences 18(2), 110–127 (1979)