アルゴリズム 60－6
（1998. 1. 22）

重みのある場合とない場合に、k 個のソートされた列に対する選択問題を解くアルゴリズム

林 達也　中野 浩嗣
名古屋工業大学
電気情報工学科

Stephan Olariu
Department of Computer Science
Old Dominion University

本論文では、重み付き選択問題を解く $O(\log n \log^* n)$ 時間の EREW-PRAM 上の仕事量最適の並列アルゴリズムを示す。そして、総データ数が n の k 個のソートされた列が与えられたときに、重みなし選択問題を解く最適な $O(k \log \frac{n}{k})$ 時間の逐次アルゴリズムをしめす。また、仕事量最適で、$O(\log k(\log^* k + \log \frac{n}{k}))$ 時間並列アルゴリズムを示す。最後に、重みあり選択問題について、$O(\log n)$ 時間の仕事量・時間最適の EREW-PRAM 上の並列アルゴリズムを示す。

Weighted and Unweighted Selection Algorithms for k Sorted Sequences

Tatsuya Hayashi, Koji Nakano
Department of Electrical and Computer Engineering
Nagoya Institute of Technology
Showa-ku, Nagoya 466, JAPAN
{hayashi,nakano}@elcom.nitech.ac.jp

Stephan Olariu
Department of Computer Science
Old Dominion University
Norfolk, Virginia 23529, USA
olariu@cs.odu.edu

Our first main contribution is to provide a novel EREW algorithm for weighted selection, running in $O(\log n \log^* n)$ time with optimal work. Our second main contribution is to propose lower bounds and matching upper bounds for selection and weighted selection in a collection A of k, $(1 \leq k \leq n)$, sorted sequences of combined length n. While $\Omega(n)$ remains a lower bound on the amount of work needed for weighted selection, unweighted selection has a lower bound of $\Omega(k \log \frac{n}{k})$. We go on to propose an optimal sequential algorithm for selection in k sorted sequences running in $O(k \log \frac{n}{k})$ time, as well as a work-optimal EREW algorithm running in $O(\log k(\log^* k + \log \frac{n}{k}))$ time. Finally, we present a work-time optimal EREW algorithm solving the weighted selection problem in k sorted sequences, running in $O(\log n)$ time whenever $k \leq \frac{n}{\log^{(O(1))} n}$.

1 Introduction

The model of computation adopted in this work is the EREW-PRAM [7]. Consider a parallel algorithm that solves an instance of size n of some problem in time $T_p(n)$ using p processors. The work $W(n)$ performed by the algorithm is the product $p \times T_p(n)$. The algorithm is termed *work-optimal* if $W(n) = \Theta(T^*(n))$, where $T^*(n)$ is the running time of the fastest *sequential* algorithm for the problem. An algorithm is termed *work-time optimal* [7] if it is work-optimal and, in addition, its running time $T_p(n)$ is best possible among the work-optimal algorithms in that model.

Given a set A of n items and an integer m, $1 \leq m \leq n$, the (unweighted) *selection problem* asks for the m-th smallest item in A. It is well-known that selection can be solved in $\Theta(n)$ sequential time [2]. By parallelizing this sequential selection algorithm, Akl [1] obtained a work-optimal EREW selection algorithm running in $O(n^\epsilon)$ time and performing $O(n)$ work, for every fixed $0 < \epsilon \leq 1$. Vishkin [9] showed that selection can be solved in $O(\log n \log\log n)$ time and $O(n)$ work. Later, Cole [4] obtained a selection algorithm running in $O(\log n \log^* n)$ EREW time and performing $O(n)$ work. To this day it is not known whether Cole's algorithm is work-time optimal.

Consider a positive number m, a set $A = \{a_1, a_2, \ldots, a_n\}$ of items, and a set of non-negative

weights $\{w_1, w_2, \ldots, w_n\}$, where w_i is the *weight* of a_i, $1 \leq i \leq n$. The *weighted selection problem* asks for the largest item a_j in A for which the total weight of the items smaller than or equal to a_j does not exceed m. It is straightforward to design an $O(n)$ time sequential algorithm for the weighted selection problem. As it turns out, the task of designing an efficient parallel algorithm for this problem is much harder. Somewhat surprisingly, none of the known parallel selection algorithms work for weighted selection. This motivated Chen *et al.* [3] to design an EREW algorithm for this problem running in $O(\log n \log\log n)$ time and using $O(n)$ work. Our first main contribution is to modify the unweighted selection EREW algorithm in [3] to solve the weighted selection problem in $O(\log n \log^* n)$ time and $O(n)$ work.

Consider a collection A of k, $(1 \leq k \leq n)$, sorted sequences of combined length n. Our second main contribution is to propose lower bounds and matching upper bounds for the problems of selection in k sorted sequences. We prove that selection in k sorted sequences has a lower bound of $\Omega(k \log \frac{n}{k})$ time and show that this lower bound is tight by exhibiting a simple selection algorithm running in $O(k \log \frac{n}{k})$ time. Clearly, our sequential algorithm translates into a selection algorithm in a sorted matrix of size $m \times n$, $(m \leq n)$, running in $O(m \log n)$ time. Thus, whenever $m \leq n^{1-\epsilon}$ for every fixed $\epsilon > 0$, our sequential algorithm matches the running time of the optimal sequential algorithm in [6]. We also propose a work-optimal EREW algorithm running in $O(\log k(\log^* k + \log \frac{n}{k}))$ time using $O(k \log \frac{n}{k})$ work.

A matrix is *sorted* if its rows and columns are independently sorted. Frederickson and Johnson[6] offered an optimal sequential selection algorithm in a sorted matrix of size $m \times n$, $(m \leq n)$, running in $\Theta(m \log \frac{n}{m})$ time. Quite recently, Shen [8] presented an EREW-PRAM algorithm for selection in a sorted matrix of the same size running in $O(\log m \log^* m(\log\log m + \log \frac{n}{m}))$ time and using $O(\frac{m}{\log m \log^* m})$ processors. Shen's algorithm is work-optimal whenever $m \log m \leq n$. When applied to a sorted matrix of size $m \times n$, $(m \leq n)$, our algorithm runs in $O(\log m(\log^* m + \log n))$ time and $O(m \log n)$ work. Thus, whenever $m \leq n^{1-\epsilon}$, our algorithm is work-optimal even for sorted matrices and runs faster than Shen's algorithm [8].

Next, we address the weighted selection problem in k sorted sequences. Although (unweighted) selection can be performed in sublinear time, $\Omega(n)$ remains a lower bound on the work needed for weighted selection. We show that for $k \leq \frac{n}{\log^{(O(1))} n}$ the weighted selection problem in k sorted sequences can be solved by a work-time optimal EREW algorithm in $O(\log n)$ time and $O(n)$ work. Since $\Omega(\log k)$ is a time lower bound for this problem, regardless of the number of processors, our algorithm is work-time optimal whenever $n^\epsilon \leq k \leq \frac{n}{\log^{(O(1))} n}$ for every fixed $\epsilon > 0$.

2 Preliminaries

Let $A = \{a_1, a_2, \ldots, a_n\}$ be a set of distinct[1] items with each a_i having a weight $w_i > 0$. Enumerate A in sorted order as $a_{l(1)} < a_{l(2)} < \cdots < a_{l(n)}$. We say that $a_{l(i)}$ is the i-th smallest item in A or that its *rank* is i. The *weighted rank* of $a_{l(i)}$ is defined to be $r(l(i)) = w_{l(1)} + w_{l(2)} + \cdots + w_{l(i)}$.

Given a positive number m, the *weighted selection* problem asks for the item $a_{l(i)}$ in A, if any, satisfying $r(l(i)) \leq m < r(l(i+1))$. If such an item $a_{l(i)}$ exists, it is termed the item of *lower weighted rank* m. The item of *upper weighted rank* m is $a_{l(i)}$ if $r(l(i)) = m$ and $a_{l(i+1)}$ otherwise. It is obvious that the task of selecting the m-th smallest item has a time lower bound of $\Omega(n)$ because, in the worst case, every item must be accessed at least once. This implies an $\Omega(n)$ lower bound for the weighted selection problem as well. The problem can be solved in $O(n)$ time as follows:

`Sequential-Weighted-Selection`(A, m)

Step 1 If $n = 1$ return 1 or $\emptyset$ depending on whether of not $w_1 \leq m$;

Step 2 If $n > 1$ compute the $\lceil \frac{n}{2} \rceil$-th smallest item of A and its weighted rank r;

Step 3 If $r = m$ then output x. If $r > m$ then remove all items larger than x and recursively find the item of lower weighted rank m among the remaining items. Otherwise, remove x and all items smaller than x and recursively find the item of lower weighted rank $(m-r)$ among the remaining items.

Since each recursive call takes linear time and the number of remaining items is reduced by half in each recursive call, we have the following result.

Lemma 2.1 *An arbitrary instance of size n of the weighted selection problem can be solved optimally in $\Theta(n)$ time.*

[1]This is not really a restriction, since we can make all the elements distinct by interpreting them as ordered pairs of the form (a_i, i).

3 A parallel algorithm for weighted selection

In this section we present a work-optimal EREW algorithm for weighted selection. This algorithm is based on the technique used by Cole's parallel unweighted selection algorithm [4]. We begin by taking note of the following naive solution.

`Naive-Weighted-Selection`(A, m)

Step 1 Sort $A = \{a_1, a_2, \ldots, a_n\}$ as $a_{l(1)} < a_{l(2)} < \cdots < a_{l(n)}$;

Step 2 Compute the prefix sums of the corresponding weights $w_{l(1)}, w_{l(2)}, \ldots, w_{l(n)}$ and determine the weighted ranks $r(l(1)), r(l(2)), \ldots, r(l(n))$;

Step 3 Find the index i satisfying $r(l(i)) \leq m < r(l(i+1))$ and return $a_{l(i)}$ as the item of lower weighted rank m.

Clearly, the complexity is dominated by Step 1 that runs in $O(\log n)$ time using n processors [5]. Thus, we have

Lemma 3.2 *An arbitrary instance of size n of the weighted selection problem can be solved in $O(\log n)$ time and $O(n \log n)$ work on the EREW.*

Similarly to the Cole's parallel unweighted selection algorithm [4], we present an approximate weighted selection algorithm. Consider the sequence defined by $t_1 = 1$ and by $t_{i+1} = 2^{t_i}$ for $i \geq 1$. Clearly, $\log n \leq t_{\log^* n} < n$. Consider the weighted selection problem with parameters (A, m), where $A = \{a_1, a_2, \ldots, a_{\frac{n}{t_i^2}}\}$. Write A in increasing order as $a_{l(1)} < a_{l(2)} < \cdots < a_{l(\frac{n}{t_i^2})}$ and let $a_{l(s)}$ be the item in A of lower weighted rank m. The following algorithm finds a good approximation of $a_{l(s)}$ in $O(\log n)$ time and $O(\frac{n}{t_i})$ work. More precisely, the algorithm returns the item $a_{l(s')}$ satisfying $s - \frac{n}{2t_i^2 t_{i+1}^2} \leq s' \leq s$.

`Approximate-Weighted-Selection`(A, m)

Step 1 If $t_i \geq \log n$, then use algorithm `Naive-Weighted-Selection` to find the exact solution and EXIT. Otherwise, execute the following steps.

Step 2 Partition A into $\frac{n}{4t_{i+1}^4 t_i^2}$ subsets $A_1, A_2, \ldots, A_{\frac{n}{4t_{i+1}^4 t_i^2}}$ each of size $4t_{i+1}^4$, such that for each j, $(1 \leq j \leq \frac{n}{4t_{i+1}^4 t_i^2})$, $A_j = \{a_{4t_{i+1}^4(j-1)+1}, \ldots, a_{4t_{i+1}^4 j}\}$. Sort each A_j as $a_{j,1} < a_{j,2} < \cdots < a_{j,4t_{i+1}^4}$ and let $w_{j,k}$ be the weight of $a_{j,k}$.

Step 3 Partition each sequence A_j into $4t_{i+1}^2$ subsets $A_{j,1}, A_{j,2}, \ldots, A_{j,4t_{i+1}^2}$ of t_{i+1}^2 items each such that $A_{j,k} = \{a_{j,4t_{i+1}^2(k-1)+1}, \ldots, a_{j,4t_{i+1}^2 k}\}$. Compute the total weight $w'_{j,k} = w_{j,4t_{i+1}^2(k-1)+1} + \cdots + w_{j,4t_{i+1}^2 k}$ of each $A_{j,k}$.

Step 4 Let $a'_{j,k}$ denote the item $a_{j,4t_{i+1}^2 k}$, that is, the last item in $A_{j,k}$ and denote its weight by $w'_{j,k}$. Run the algorithm recursively with parameters $A' = \bigcup_{1 \leq j \leq \frac{n}{4t_{i+1}^4 t_i^2}, 1 \leq k \leq 4t_{i+1}^2} a'_{j,k}$ and m, and return the resulting item as $a_{l(s')}$.

Note that if $t_i \geq \log n$, Step 1 runs in $O(\log n)$ time with $O(\frac{n \log n}{t_i^2}) \leq O(\frac{n}{t_i})$ work. If $t_i < \log n$, Step 2 runs in $O(\log(4t_{i+1}^4)) = O(t_i)$ time with $O(4t_{i+1}^4 \log(4t_{i+1}^4)) = O(t_{i+1}^4 t_i)$ work for each subset A_j. Thus, the total work in Step 2 is $O(t_{i+1}^4 t_i) \times \frac{n}{4t_{i+1}^4 t_i^2} = O(\frac{n}{t_i})$. Step 3 takes $O(\log t_{i+1}^2) = O(t_i)$ time and $O(\frac{n}{t_i^2})$ work. In Step 4, A' has $4t_{i+1}^2 \times \frac{n}{4t_{i+1}^4 t_i^2} = \frac{n}{t_{i+1}^2 t_i^2} < \frac{n}{t_{i+1}^2}$ items. Hence, the next recursive call in Step 4 needs $O(t_{i+1})$ time and $O(\frac{n}{t_{i+1}})$ work. Thus, the total time in all the recursive calls is $O(t_i) + O(t_{i+1}) + \cdots + O(\log n) = O(\log n)$ and the work is $O(\frac{n}{t_i}) + O(\frac{n}{t_{i+1}}) + \cdots + O(\frac{n}{\log n}) = O(\frac{n}{t_i})$.

Let $a_{l(s)}$ be the item of lower weighted rank m in A and let $a_{l(s')}$ be the item returned by the algorithm. Assume that $a_{l(s')}$ is the item in A' of *exact* lower weighted rank m and refer to Figure 1. Here, the shaded part illustrates the items in A smaller than or equal to $a_{l(s)}$. The dark circles denote items in A' smaller than or equal to $a_{l(s')}$. It is clear that, in each A_i, the "error" is bounded by t_{i+1}^2. Thus, the total error does not exceed $t_{i+1}^2 \times \frac{n}{4t_{i+1}^4 t_i^2} = \frac{n}{4t_{i+1}^2 t_i^2}$. In other words, $a_{l(s')}$ is the item of lower weighted rank m with an error of at most $\frac{n}{4t_{i+1}^2 t_i^2}$. Now, assume that Step 4 finds the weighted m-th item $a_{l(s')}$ of A' with error e_{i+1}. With this assumption, $a_{l(s')}$ is the item of lower weighted rank m with error at most $t_{i+1}^2 \times (\frac{n}{4t_{i+1}^4 t_i^2} + e_{i+1}) = \frac{n}{4t_{i+1}^2 t_i^2} + t_{i+1}^2 e_{i+1}$. Thus, $e_i = 0$ if $i = \log^* n$ and $e_i \leq \frac{n}{4t_{i+1}^2 t_i^2} + t_{i+1}^2 e_{i+1}$ for $1 \leq i \leq \log^* n - 1$. It is easy to show that $e_i \leq \frac{n}{2t_i^2 t_{i+1}^2}$ for $1 \leq i \leq \log^* n - 1$.

Lemma 3.3 *With a collection A of $\frac{n}{t_i^2}$ weighted items and a positive number m as input, algorithm* `Approximate-Weighted-Selection` *returns an approximation to the item of lower weighted rank m with error at most $\frac{n}{2t_i^2 t_{i+1}^2}$ in $O(\log n)$ time and $O(\frac{n}{t_i})$ work on the EREW.*

With minimal modifications the above algorithm produces an approximation to the item of upper

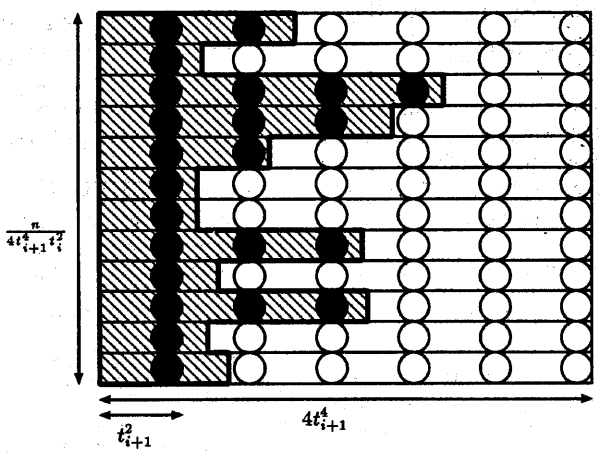

Figure 1: *Illustrating the items smaller than* $a_{l(s)}$.

weighted rank m as well. Specifically, if $a_{l(z)}$ is the item of *exact* upper weighted rank m, we obtain an item $a_{l(z')}$ such that $z \leq z' \leq z + \frac{n}{2t_i^2 t_{i+1}^2}$. Observe that, in the previous notation, $s' < s \leq z < z'$ and so we can eliminate all the items outside of these bounds. Using this fact, Cole's parallel unweighted selection algorithm [4] repeats the approximate unweighted selection algorithm to remove items that cannot be the item to be selected. More precisely, in the i-th iteration $(1 \leq i \leq \log^* n)$, the number of items is reduced from $\frac{n}{t_i^2}$ to $\frac{n}{t_{i+1}^2}$. In the same manner, by repeating `Approximate-Weighted-Selection`, we can find the exact weighted item. Due to stringent page limitations, the details are omitted. The reader should refer to [4] for this iteration technique. Thus, we have the following result.

Theorem 3.4 *An arbitrary instance of size n of the weighted selection problem can be solved in $O(\log n \log^* n)$ time and $O(n)$ work on the EREW.*

4 Unweighted selection in k sorted sequences

Let A be a collection of k sorted sequences $A_1, A_2, \ldots, A_k$ of combined length n. The problem is to return the m-th smallest item in A. To prove a lower bound on the computing time for selection in k sorted sequences, we introduce the *checking problem* for selection as follows: given k sorted sequences $A_1, A_2, \ldots, A_k$, and an item x, decide whether x is the m-th smallest item in the collection. Clearly, the checking problem is not harder than selection and, consequently, any lower bound for the checking problem is also a lower bound for selection.

If the rank of x with respect to some A_i is not known, then we cannot decide whether the rank of x is m. Thus, the rank of x with respect to every A_i has to be computed. To compute the rank x with respect to A_i, $\Omega(\log |A_i|)$ comparisons are necessary. If each $|A_i|$ has $\frac{n}{k}$ items then, overall, $\Omega(k \log \frac{n}{k})$ time is required to check whether x is the m-th item. Thus, we have

Lemma 4.5 *Any algorithm that correctly solves the selection problem in k sorted sequences of combined length n requires $\Omega(k \log \frac{n}{k})$ time in the worst case.*

4.1 An optimal sequential algorithm

In this subsection we design an sequential algorithm matching the lower bound we just proved. Our idea is simple: in each iteration we remove a constant fraction of the number of items that cannot be the m-th smallest item. This removal is repeated $O(\log \frac{n}{k})$ times, until the number of items is reduced to $O(k)$, at which moment we use the selection algorithm of [2].

To remove a constant fraction of the remaining items in $O(k)$ time, we develop a novel sampling technique. For simplicity, assume that $m \geq \frac{n}{2}$. Select a sample $B = \{b_1, b_2, \ldots, b_k\}$ by retaining in each A_i the $\lfloor \frac{|A_i|}{4} \rfloor$-th smallest item b_i. (In case A_i has fewer than four items, we

set $b_i = +\infty$.) The item b_i partitions A_i into two subsequences $A_i^- = \{x \in A_i \mid x \leq b_i\}$ and $A_i^+ = \{x \in A_i \mid x > b_i\}$. With every sample item b_i in B we associate the weight $w_i = |A_i|$. Notice that the total weight of B is n. By reindexing $A_1, A_2, \ldots, A_k$ for the sorted order of B we assume without loss of generality that $b_1 < b_2 < \cdots < b_k$.

Next, having selected the item b_s of upper weighted rank $\frac{n}{4}$ in B, we proceed to partition B into two subsets $B^- = \{b_1, b_2, \ldots, b_s\}$, and $B^+ = \{b_{s+1}, b_{s+2}, \ldots, b_k\}$. We now partition A into six subsets: $A^{--} = A_1^- \cup A_2^- \cup \cdots \cup A_{s-1}^-$, $A^{-+} = A_1^+ \cup A_2^+ \cup \cdots \cup A_{s-1}^+$, $A^{+-} = A_{s+1}^- \cup A_{s+2}^- \cup \cdots \cup A_k^-$, $A^{++} = A_{s+1}^+ \cup A_{s+2}^+ \cup \cdots \cup A_k^+$, A_s^-, and A_s^+, as illustrated in Figure 2.

Lemma 4.6 *If $m \geq \frac{n}{2}$ then $A^{--} \cup A_s^-$ cannot contain the m-th smallest item in A.*

Proof. Since b_s is the item of upper weighted rank $\frac{n}{4}$ in B, $A^{--} \cup A^{-+}$ contains at most $\lfloor \frac{n}{4} \rfloor$ items. Also, since each A_i^- has $\lfloor \frac{|A_i|}{4} \rfloor$ items of A_i, $A^{--} \cup A_s^- \cup A^{+-}$ has at most $\lfloor \frac{n}{4} \rfloor$ items. Thus, $A^{--} \cup A_s^- \cup A^{+-} \cup A^{-+}$ has less than $\frac{n}{2}$ items, and $A^{++} \cup A_s^+$ has more than $\frac{n}{2}$ items. Further, every item in $A^{++} \cup A_s^+$ is larger than b_s. Hence, the m-th smallest item must be larger than b_s. Also, no item in $A^{--} \cup A_s^-$ is larger than b_s. Thus, $A^{--} \cup A_s^-$ cannot contain the m-th smallest item in A. □ Lemma 4.6 guarantees that all the items in $A^{--} \cup A_s^-$ can be removed from further consideration. As it turns out, $A^{--} \cup A_s^-$ has approximately $\frac{n}{16}$ items. Thus, we can remove a constant fraction of the remaining items. Note that if $m < \frac{n}{2}$, we can remove items in a symmetric way.

Lemma 4.7 *$A^{--} \cup A_s^-$ contains at least $\frac{n-12k}{16}$ items.*

Proof. Since b_s is the item of upper weighted rank $\frac{n}{4}$ in B, $A^{--} \cup A^{-+} \cup A_s^- \cup A_s^+ = A_1 \cup A_2 \cup \cdots \cup A_s$ contains at least $\frac{n}{4}$ items. Since each b_i is the $\lfloor \frac{|A_i|}{4} \rfloor$-th smallest item in A_i, we have $|A^{--} \cup A_s^-| = \sum_{i=1}^{s} \lfloor \frac{|A_i|}{4} \rfloor \geq \frac{|\bigcup_{i=1}^{s} A_i| - 3s}{4} \geq \frac{n-12s}{16} \geq \frac{n-12k}{16}$. □

Lemmas 4.6 and 4.7 suggest the following selection algorithm in k sorted sequences.

`Selection-in-Sorted-Sequences`(A, m)

Step 1 Compute the total number of remaining items and check whether it is less than or equal to $13k$.

Step 2 If at most $13k$ items remain, find m-th smallest item in $O(k)$ time and EXIT. Otherwise, execute the following steps.

Step 3 Check whether or not $m \geq \frac{n}{2}$. Assume, without loss of generality, that $m \geq \frac{n}{2}$.

Step 4 Select a sample B by retaining the $\lfloor \frac{|A_i|}{4} \rfloor$-th smallest item in each A_i .

Step 5 Find b_s, the item of upper weighted rank $\frac{n}{4}$ in B.

Step 6 Remove all items in $A^{--} \cup A_s^-$.

Step 7 Compute the number m' of items removed in Step 6.

Step 8 Recursively find the $(m - m')$-th smallest item in the remaining items.

The correctness follows directly from Lemma 4.6. Initially, the sequences $A_1, A_2, \ldots, A_k$ are stored in an array $a[1..n]$. For definiteness, assume that the items in A_1 are $a[1], a[2], \ldots, a[|A_1|]$, the items in A_2 are $a[|A_1|+1], a[|A_1|+2], \ldots a[|A_1|+|A_2|]$, and so on. Since the remaining items in each A_i form an interval, we use two arrays $u[1..k]$ and $v[1..k]$ such that for every i, $(1 \leq i \leq k)$, the items remaining in A_i are $a[u[i]], a[u[i]+1], \ldots, a[v[i]]$. The arrays u and v are updated in Step 6.

Observe that the number of items in A_i is $v[i] - u[i] + 1$. Thus, the total number n' of remaining items can be computed in $O(k)$ time. After that, checking whether $n' \leq 13k$ can be done in $O(1)$ time. Therefore, Step 1 runs in $O(k)$ time. In Step 2 we use `Sequential-Weighted-Selection`. By Lemma 2.1 this takes $O(k)$ time. Since the total number n' of items is computed in Step 1, Step 3 takes $O(1)$ time. Since $a[u[i] + \lfloor \frac{v[i]-u[i]+1}{4} \rfloor]$ is the $\lfloor \frac{|A_i|}{4} \rfloor$-th smallest item in each A_i, Step 4 takes $O(k)$ time. Step 5 uses `Sequential-Weighted-Selection` and takes $O(k)$ time. Finding items in B smaller than or equal to b_s can be done in $O(k)$ time in the obvious way. After that, to remove all items in $A^{--} \cup A_s^-$ we update the arrays u and v in $O(k)$ time. For example, $u[1]$ is updated by $u[1] + \lfloor \frac{v[1]-u[1]+1}{4} \rfloor + 1$. Thus, Step 6 takes $O(k)$ time. Step 7 runs in $O(k)$ time. Thus, all the steps can be performed in $O(k)$ time.

To estimate the depth of the recursion, let $S(t)$ denote the number of remaining items in the t-th recursion level. Clearly, $S(0) = n$. Lemma 4.7 implies that for $t \geq 1$, $S(t) \leq S(t-1) - \frac{S(t-1)-12k}{16}$. The solution of this recurrence satisfies for all $t \geq 0$ $S(t) \leq (\frac{15}{16})^t n + 12k$. The recursion terminates when $S(t) \leq 13k$. Hence, the depth of the recursion is the smallest T satisfying $(\frac{15}{16})^T n + 12k \leq 13k$ implying that $T = O(\log \frac{n}{k})$. To summarize, we proved the following important result.

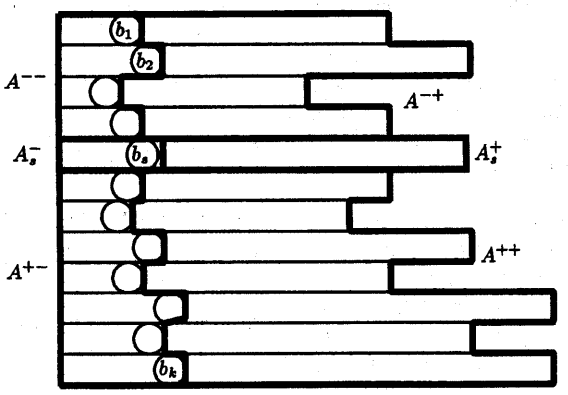

Figure 2: *Illustrating the partitioning of A.*

Theorem 4.8 *The task of selecting the m-th smallest item in a collection of k sorted sequences of combined length n can be performed in $O(k \log \frac{n}{k})$ time. Furthermore, this is the best possible.*

4.2 A work-optimal parallel algorithm

We begin by discussing a simple parallelization of the sequential algorithm of the previous subsection. By Theorem 3.4, Steps 2 and 5 can be implemented in $O(\log k \log^* k)$ time and $O(k)$ work. All the other steps can be implemented to run in $O(\log k)$ time and $O(k)$ work in the obvious way. Thus, each recursion level takes $O(\log k \log^* k)$ time and $O(k)$ work. Since the depth of the recursion is $O(\log \frac{n}{k})$, the algorithm can be implemented to run in $O(\log k \log^* k \log \frac{n}{k})$ time and $O(k \log \frac{n}{k})$ work.

We now show that with a careful implementation the above algorithm can run in $O(\log k(\log^* k + \log \frac{n}{k}))$ time, while performing the same amount of work. The first idea is to use `Approximate-Weighted-Selection` in Step 2 instead of the algorithm for Theorem 3.4. Indeed, if we use algorithm `Approximate-Weighted-Selection` in Step 2, and if we can still guarantee that a constant fraction of items are removed in each recursive call, the computing time remains the same. However, we cannot ensure this fact as we are going to show. Lemma 3.3 guarantees that, for every fixed $c > 0$, an item with an error bound of at most $\frac{n}{c}$ can be found in $O(\log n)$ time and $O(n)$ work. Now, in Step 5, instead of finding the item of upper weighted rank $\frac{n}{4}$ we find the item of upper weighted rank $\frac{n}{4}$ with an error of, say, at most $\frac{k}{8}$. Let b_s be the (unknown) item of upper weighted rank $\frac{n}{4}$. Further, let $b_1 < b_2 < \cdots < b_{s-1}$ be the items in B smaller than b_s. Then, in the worst case, the approximate item computed by `Approximate-Weighted-Selection` is $b_{s-\frac{k}{8}}$. Let us estimate the number of items that will be removed. Imagine that the input is partitioned into six subsets, with respect to $b_{s-\frac{k}{8}}$, as described in the previous subsection. If each of $A_1, A_2, \ldots, A_{s-\frac{k}{8}}$ has few items and $A_{s-\frac{k}{8}+1}$, $A_{s-\frac{k}{8}+2}$, ..., A_s have many items, we cannot guarantee that the set $A^{--} \cup A^-_{s-\frac{k}{8}}$ contains a constant fraction of the remaining items.

However, in case none of the sequences A_i has too many items, we can guarantee that $A^{--} \cup A^-_{s-\frac{k}{8}}$ contains a constant fraction of n. To guarantee this, each sequence is temporary partitioned into subsequences. More precisely, each A_i is partitioned into $\lceil \frac{k|A_i|}{n} \rceil$ sorted sequences, each of size at most $\lfloor \frac{n}{k} \rfloor$. For example, $A_1 = \{a_1, a_2, \ldots\}$ is partitioned into subsequences $A_{1,1} = \{a_1, a_2, \ldots, a_{\lfloor \frac{n}{k} \rfloor}\}$, $A_{1,2} = \{a_{\lfloor \frac{n}{k} \rfloor + 1}, a_{\lfloor \frac{n}{k} \rfloor + 2}, \ldots, a_{2\lfloor \frac{n}{k} \rfloor}\}$, ..., and the last sequence may have fewer than $\lfloor \frac{n}{k} \rfloor$ items. After partitioning, the input is partitioned into at most $2k$ subsequences, each of size at most $\lfloor \frac{n}{k} \rfloor$. Note that the items need not be moved. The only thing we need is to extend the arrays $u[1..k]$ and $v[1..k]$ into $u'[1..2k]$ and $v'[1..2k]$. For example, if A_1 is partitioned as discussed above then $u'[1] = 1, v'[1] = \lfloor \frac{n}{k} \rfloor$, $u'[2] = \lfloor \frac{n}{k} \rfloor + 1$, $v'[2] = 2\lfloor \frac{n}{k} \rfloor$, and so on. This extension can be done in $O(\log k)$ time and $O(k)$ work using an optimal prefix-sums algorithm [7]. After this partitioning, Steps 1 to 4 of `Selection-in-Sorted-Sequences` are executed for the resulting subsequences. In Step 5, the

item of approximate upper weighted rank $\frac{n}{4}$ in B with error at most $\frac{k}{8}$ is computed by using `Approximate-Weighted-Selection`. Since B has at most $2k$ items, this can be done in $O(\log k)$ time and $O(k)$ work. Let $b_{s'}$ be the item returned. The items in B smaller than $b_{s'}$ are retained in the same way as in Step 6. After that, we update the arrays $u[1..n]$ and $v[1..n]$ to remove the items in $A^{--} \cup A_{s'}$. Notice that we do not update $u'[1..n]$ and $v'[1..n]$. It is possible that B has several items in A_i and two or more of them are smaller than $b_{s'}$. In this case, each $u[i]$ is updated by the maximum of the smaller items for A_i. For example, assume that both $A_{1,1}$ and $A_{1,2}$ have an item in B smaller than $b_{s'}$, and $A_{1,3}$ does not have such an item. Then, $u[1]$ is updated by $u'[2] + \lfloor \frac{v'[2]-u'[2]}{4} \rfloor$. Once this is done, the algorithm is executed recursively for the updated arrays $u[1..n]$ and $v[1..n]$.

The partitioning we discussed increases the number of sequences to at most $2k$, but ensures that each sorted sequence has at most $\lfloor \frac{n}{k} \rfloor$ items. Thus, we can ensure that at least $\frac{n-12k}{16} - \frac{k}{8} \times \frac{\frac{n}{k}}{4} = \frac{n-24k}{32}$ items are removed in Step 6, confirming that the depth of the recursion is still $O(\log \frac{n}{k})$. Note that in Step 2, if the number of the remaining item is at most $25k$ then the weighted selection algorithm for Theorem 3.4 is used. In this case, the weighted selection algorithm runs in $O(\log k \log^* k)$ and $O(k)$ work. Thus, we have

Theorem 4.9 *The task of selecting the m-th smallest item in a collection of k sorted sequences of combined length n can be performed in $O(\log k(\log^* k + \log \frac{n}{k}))$ time and $O(k \log \frac{n}{k})$ work on the EREW. Furthermore, this is work-optimal.*

5 Weighted selection in k sorted sequences

In this section we present a parallel algorithm for weighted selection in k sorted sequences $A_1, A_2, \ldots, A_k$, of combined length n. Since every weight must be accessed in the worst case, the weighted selection problem needs $\Omega(n)$ sequential time.

In the i-th iteration of the algorithm for Theorem 3.4, the number of items is reduced from $\frac{n}{t_i^2}$ to $\frac{n}{t_{i+1}^2}$. Thus, we have the following consequence.

Corollary 5.10 *Given $\frac{n}{t_i^2}$ weighted items and a number $m > 0$, the items of lower (resp. upper) weighted rank m can be found in $O((\log^* n - i + 1) \log n)$ time and $O(n)$ work.*

Using the above as a subroutine, the following algorithm finds the item of lower weighted rank m in $\frac{n}{t_i^2}$ sorted sequences $A_1, A_2, \ldots, A_{\frac{n}{t_i^2}}$ of total length n.

`Weighted-Selection-in-Sorted-Sequences`(A, m)

Step 1 Partition each sequence into subsequences of size t_i^2. For example, A_1 is partitioned into subsequences $A_{1,1} = \{a_1, a_2, \ldots, a_{t_i^2}\}$, $A_{1,2} = \{a_{t_i^2+1}, a_{t_i^2+2}, \ldots, a_{2t_i^2}\}$, and so on. Compute the total weight $w_{i,j}$ of each subsequence $A_{i,j}$ and let $a_{i,j}$ denote the largest item in $A_{i,j}$. Let A' be the set of all the items $a_{i,j}$, each with weight $w_{i,j}$.

Step 2 Compute the items of lower and upper weighted rank m in A'.

Step 3 From each sequence A_i, remove all items smaller than the item of lower weighted rank m. Compute the total weight m' of the items removed. Remove all items larger than the item of upper weighted rank m.

Step 4 Return the item of lower weighted rank $(m - m')$ among the remaining items.

Step 1 partitions the input into at most $\frac{n}{t_i^2} + \frac{n}{t_i^4} \leq \frac{2n}{t_i^2}$ subsequences. Consequently, A' has at most $\frac{2n}{t_i^2}$ items. Corollary 5.10 guarantees that Step 2 runs in $O((\log^* n - i + 1) \log n)$ time and $O(n)$ work. Step 3 takes $O(\log n)$ time and $O(n)$ work. Since no item in B is between the items of lower and upper weighted rank m, each sequence A_i has at most t_i^2 remaining items. Hence, altogether, at most $t_i^2 \times \frac{n}{t_i^4} = \frac{n}{t_i^2}$ items remain. Thus, Step 4 can be performed in $O((\log^* n - i + 1) \log n)$ time and $O(n)$ work. Thus, the item of lower weighted rank m can be found in $O((\log^* n - i + 1) \log n)$ and $O(n)$ work. To summarize, we state the following result.

Theorem 5.11 *Given $\frac{n}{t_i^2}$ sorted sequences with weighted items of combined length n and a positive number m, the items of lower and upper weighted rank m can be found in $O((\log^* n - i + 1) \log n)$ time and $O(n)$ work.*

Let $t_i = \log^{(O(1))} n$. As a corollary, we have,

Corollary 5.12 *Given k sorted sequences with weighted items of combined length n and a positive number m, the items of lower and upper weighted rank m can be found in $O(\log n)$ time and $O(n)$ work, whenever $k \leq \frac{n}{\log^{(O(1))} n}$.*

Hence, for example, if subsequences of $\log^{(O(1))} n$ items each are sorted locally, then the weighted

selection can be done in $O(\log n)$ time and $O(n)$ work. Further, in this case, this algorithm is work-time optimal.

Acknowledgment

This work is supported in part by NSF grant CCR-9522093, by ONR grant N00014-97-1-0526, by Grant-in-Aid for Encouragement of Young Scientists (09780262) from Ministry of Education, Science, Sports, and Culture of Japan, and by a grant from the Hori Information Science Promotion Foundation.

References

[1] S. G. Akl. An optimal algorithm for parallel selection. *Information Processing Letters*, 19(1):47–50, July 84.

[2] M. Blum, R.W. Floyd, V.R. Pratt, R.L. Rivest, and R.E. Tarjan, Time bounds for selection. *Journal of Computer and System Sciences*, (7):448–461, 1973.

[3] D. Z. Chen, W. Chen, K. Wada, and K. Kawaguchi. Parallel algorithms for partitioning sorted sets and related probles. In *Proc. 4th European Symposium on Algorithms (ESA'96, LNCS 1136)*, pages 234–245, 1996.

[4] R. Cole. An optimally efficient selection algorithm. *Information Processing Letters*, 26:295–299, January 1987.

[5] R. Cole, Parallel merge sort. *SIAM Journal on Computing*, 17:770–785, 1988.

[6] G. N. Frederickson and D. B. Johnson, Generalized selection and ranking: sorted matrices, *SIAM Journal on Computing*, 13:14–30, 1984.

[7] J. JáJá. *An Introduction to Parallel Algorithms.* Addison-Wesley, Reading, MA, 1992.

[8] H. Shen, Generalized parallel selection in sorted matrices, *Proc. Eight IEEE Symposium on Parallel and Distributed Systems*, October 1996, 281–285.

[9] U. Vishkin. An optimal parallel algorithm for selection. In *Advances in Computing Research.* JAI Press Inc., Greenwich CT, 1987.