The Complexity of Computing the k-ary Composition of a Binary Associative Operator

Gerth Stølting Brodal
Sven Skyum

BRICS Report Series RS-96-42

ISSN 0909-0878 November 1996

The Complexity of Computing the k-ary Composition of a Binary Associative Operator[*]

Gerth Stølting Brodal[†] and Sven Skyum

BRICS[‡]

Department of Computer Science, University of Aarhus

Ny Munkegade, DK-8000 Århus C, Denmark

{gerth,sskyum}@brics.dk

November 27, 1996

Abstract

We show that the problem of computing all contiguous k-ary compositions of a sequence of n values under an associative and commutative operator requires $3\frac{k-1}{k+1}n - \mathrm{O}(k)$ operations.

For the operator max we show in contrast that in the decision tree model the complexity is $\left(1+\Theta(1/\sqrt{k})\right)n - \mathrm{O}(k)$. Finally we show that the complexity of the corresponding on-line problem for the operator max is $\left(2-\frac{1}{k-1}\right)n - \mathrm{O}(k)$.

[*]This work was partially supported by the ESPRIT Long Term Research Program of the EU under contract #20244 (ALCOM-IT).

[†]Supported by the Danish Natural Science Research Council (Grant No. 9400044).

[‡]**B**asic **R**esearch **i**n **C**omputer **S**cience, Centre of the Danish National Research Foundation.

Introduction

Given a sequence of values $(x_1, x_2, \ldots, x_n)$ from a universe U and an associative binary operator $\oplus$, we consider the problem of computing all k-ary compositions of contiguous subsequences of length k. More formally, the problem of computing $x_i \oplus x_{i+1} \oplus \cdots \oplus x_{i+k-1}$ for $i = 1, 2, \ldots, n-k+1$.

Cooper and Kitchen presented a solution to the problem in [3], which requires $3\frac{k-1}{k+1}n$ $\oplus$-operations. We show in Section 1 that this is optimal even if $\oplus$ is commutative as well.

The solution has applications to e.g. the computer vision task of region dilation where a run-length encoding of a region can be grown vertically by or-ring together k-tuples of rows of the region (see [3]).

For special operators the k-ary composition problem is simple cases of general problems. For instance, if the universe is the booleans and the operator is disjunction, the general problem is the *boolean sum problem* [1, 7, 8]. This is the problem of computing n outputs from n inputs where each output is a disjunction of a subset of the inputs. For the general problem the number of operations might be $\Omega(n^2/\log n)$ [8].

In Section 2 we examine how many comparisons are required to solve the problem for the operator max in the decision tree model. We show that the complexity is $\left(1 + \Theta(1/\sqrt{k})\right) n - \mathrm{O}(k)$. The problem is a special case of the *set-maxima problem* [2, 4] where the problem is to compute maxima for n arbitrary subsets of the input. It is open whether subsets exist such that the complexity of the problem is $\Omega(n \log n)$. Efficient solutions to other special cases than the one dealt with in this paper have been given. In [2] an $\mathrm{O}(n)$ algorithm is given for the case where the sets are the hyperplanes in a projective geometry. For the general set-maxima problem an optimal (within a constant factor) randomised algorithm with expected complexity $\mathrm{O}(n)$ was presented in [4].

Finally, the corresponding on-line problem of computing maxima is considered in Section 3. For the on-line problem it is required that the set maxima are computed from left to right. That is, all maxima of k consecutive elements in $\{x_1, x_2, \ldots, x_{i-1}\}$ are computed before reading x_i. We show that the on-line complexity is $\left(2 - \frac{1}{k-1}\right) n - \mathrm{O}(k)$.

1 The general problem

In this section we consider how many $\oplus$-operations are required to compute the k-ary compositions. An upper bound was given in [3]. We prove that the bound achieved in [3] is optimal even if $\oplus$ is commutative as well as associative.

Let $\oplus$ be commutative as well as associative and let $(x_1, x_2, \ldots, x_n)$ be a sequence of n values from a universe U. For a subset $M \subseteq \{1, 2, \ldots, n\}$, let S_M be the composition $\bigoplus_{i \in M} x_i$. The problem is to compute

$$S_{\{1,2,\ldots,k\}}, S_{\{2,3,\ldots,k+1\}}, \ldots, S_{\{n-k+1,n-k+2,\ldots,n\}}.$$

We denote the number of $\oplus$-operations required by $\mathrm{A}(n, k)$.

We need the following notion of a computation graph and a combinatorial lemma about them.

Definition 1.1 *A directed acyclic graph $G = (V, E)$ is a* computation graph *if V consists of three kinds of nodes:* input, computation *and* output nodes*. Input nodes have indegree 0 while computation and output nodes have indegree 2. Output nodes have outdegree 0.*

Lemma 1.1 *Let $G = (V, E)$ be a computation graph with $r + 1$ input nodes $\{v_0, v_1, \ldots, v_r\}$ and t output nodes such that*

- *From any input node there is a path to some output node and*
- *to any output node there is a path from v_0.*

Then $|V| \geq 2r + 1$.

Proof: Let $M \subseteq V$ be the set of nodes on paths from v_0 to output nodes. To connect the remaining input nodes to output nodes there must be at least $\max\{r - (|M| - 1), 0\}$ non-input nodes outside M since each node in M, except v_0, has at least one of its predecessors from M. In total we get that $|V| \geq r + |M| + \max\{r - |M| + 1, 0\} \geq 2r + 1$. □

Definition 1.2 *Let $\mathcal{A}$ be an algorithm computing*

$$S_{\{1,2,\ldots,k\}}, S_{\{2,3,\ldots,k+1\}}, \ldots, S_{\{n-k+1,n-k+2,\ldots,n\}}.$$

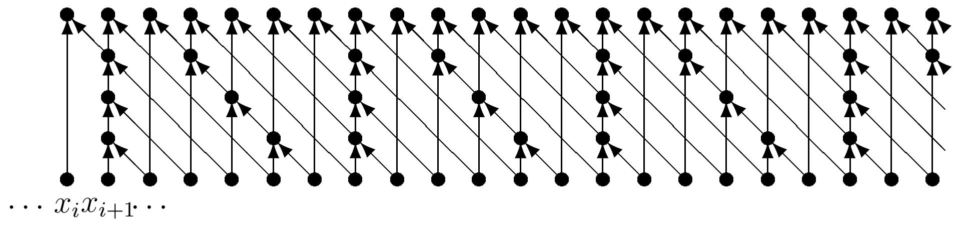

Figure 1: The computation graph for the algorithm by Cooper and Kitchen [3].

The computation graph for $\mathcal{A}$ *is the graph* $G_{\mathcal{A}} = (V_{\mathcal{A}}, E_{\mathcal{A}})$ *such that for each* S_M *computed by* $\mathcal{A}$ *there is a corresponding node* v_M*. If* $\mathcal{A}$ *computes* S_M *as* $S_{M_1} \oplus S_{M_2}$ *then there are directed edges from* v_{M_1} *and* v_{M_2} *to* v_M*.*

If the algorithm is optimal, the output nodes are exactly

$$v_{\{1,2,\ldots,k\}}, v_{\{2,3,\ldots,k+1\}}, \ldots, v_{\{n-k+1,n-k+2,\ldots,n\}}$$

corresponding to the values which are to be computed.
The computation graph for the algorithm in [3] is shown in Figure 1.

Theorem 1.2 $3\frac{k-1}{k+1}n - \mathrm{O}(k) \le \mathrm{A}(n,k) \le 3\frac{k-1}{k+1}n.$

Proof: The upper bound was proved in [3].
Now let $\mathcal{A}$ be an optimal algorithm computing

$$S_{\{1,2,\ldots,k\}}, S_{\{2,3,\ldots,k+1\}}, \ldots, S_{\{n-k+1,n-k+2,\ldots,n\}}$$

and $G_{\mathcal{A}}$ its computation graph.
Let $1 \le b \le n+1-2k$ and consider a *block* B of nodes in $V_{\mathcal{A}}$:

$$B = \{v_M \in V_{\mathcal{A}} \mid b \le \min M \le b+k\}.$$

We will show that $|B| \ge 4k-2$. The theorem follows since $V_{\mathcal{A}}$ can be split into $\lfloor \frac{n}{k+1} \rfloor$ disjoint blocks and at least $3k-3$ of the nodes in each block are computation nodes.

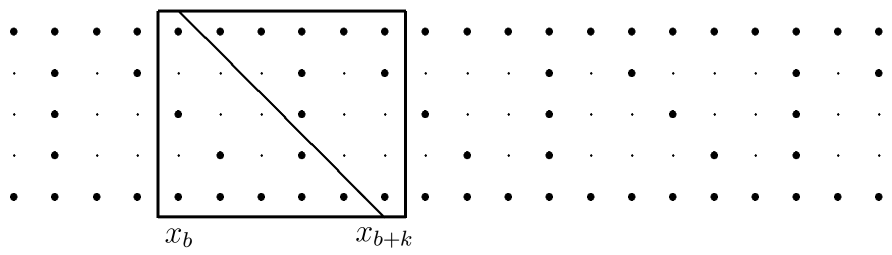

Figure 2: The division of a block of the computation graph in Figure 1 into "triangles".

Notice that if $(v_M, v_{M'}) \in E_{\mathcal{A}}$ then

$$\min M' \leq \min M \leq \max M \leq \max M'. \tag{1}$$

The block B is divided into two "triangles" L and R (see Figure 2 for an example).

$$\begin{aligned} L &= \{v_M \in V_{\mathcal{A}} \mid b \leq \min M \leq \max M < b + k\}, \\ R &= \{v_M \in V_{\mathcal{A}} \mid b < \min M \leq b + k \leq \max M\}. \end{aligned} \tag{2}$$

In L there is a path from each of the k input nodes to the output node in L. Because of (1) above, all those paths lie inside L. Consequently $|L| \geq 2k - 1$. It remains to be proven that $|R| \geq 2k - 1$ as well. Let $G_R = (V_R, E_R)$ be a computation graph with $V_R = R \cup R'$ where

$$R' = \{v_M \in V_{\mathcal{A}} \setminus R \mid \exists v_{M'} \in R : (v_M, v_{M'}) \in E_{\mathcal{A}}\}$$

and $E_R = E_{\mathcal{A}} \cap (V_R \times V_R)$.
R' contains at least $k-1$ nodes from L since for each i in $\{b+1, b+2, \ldots, b+k-1\}$ there is a path from $v_{\{i\}}$ (in L) to the output node $v_{\{i,i+1,\ldots,i+k-1\}}$ (in R) through nodes v_M with $\min M = i$. R' also contains at least $k-1$ nodes outside L since for each j in $\{b+2, b+3, \ldots, b+k\}$ there is a path from $v_{\{j\}}$ to the output node $v_{\{j,j+1,\ldots,j+k-1\}}$ (in R) through nodes v_M (outside L) with $\max M = j + k - 1 > b + k$. By identifying v_0 in Lemma 1.1 with $v_{\{b+k\}}$ and $v_1, v_2, \ldots, v_r$ with R' it follows that $|V_R| \geq 2|R'| + 1$. Now $|R'| \geq 2(k-1)$ implies that $|R| = |V_R| - |R'| \geq |R'| + 1 \geq 2k - 1$ as was to be proven. □

2 Set-maxima

In this section we examine the complexity of the problem for a constant k and a sequence $(x_1, x_2, \dots, x_n)$ to compute the *set-maxima*

$$\max\{x_i, x_{i+1}, \dots, x_{i+k-1}\}$$

for $i = 1, 2, \dots, n-k+1$ [2, 4].

If the complexity is measured as the number of max operations required we already know the complexity from the previous section. We will use the model commonly used when measuring maxima problems, namely the decision tree model. That is, we count the number of comparisons needed to solve the problem at hand. Let $\mathrm{C}(n, k)$ be the number of comparisons needed to solve the set-maxima problem in the worst case.

For simplicity, we assume throughout this section that no two elements in the sequence $(x_1, x_2, \dots, x_n)$ are equal. Otherwise the ordering $\prec$ can be used where $\prec$ is defined by $(x_i \prec x_j) \Leftrightarrow [(x_i < x_j) \vee ((x_i = x_j) \wedge (i < j))]$. We also assume $(x_1, x_2, \dots, x_n)$ to be given and fixed, so we will not continuously refer to it.

We will repeatedly build lists of maxima for prefixes and suffixes for the various subsequences of the given sequence. The following notions turn out to be helpful, expressing what is going on.

Definition 2.1 Pref *is the function that maps an interval* $[l, u]$ *to the sorted list of indices* $(i_1, i_2, \dots, i_m)$ *for maxima for prefixes of* $(x_l, x_{l+1}, \dots, x_u)$*. Formally* Pref *is defined as:*

$$\mathrm{Pref}([l, u]) = (i_1, i_2, \dots, i_m) \text{ where } l = i_1 < i_2 < \cdots < i_m \le u \text{ and}$$

$$i \in \{i_1, i_2, \dots, i_m\} \Leftrightarrow x_i = \max\{x_l, x_{l+1}, \dots, x_i\}.$$

In analogy Suff *is the function that maps an interval to the list of maxima for suffixes.*

We call the problem of computing both Pref and Suff for an interval for the *hump problem*. Let $H(n)$ be the number of comparisons needed to compute $\mathrm{Pref}([1, n])$ and $\mathrm{Suff}([1, n])$.

Another problem related to the set-maxima problem is the *augmented staircase* problem. This is the problem of computing Pref$([1,n])$ and in addition the second largest element. Let $S(n)$ be the number of comparisons needed to solve the augmented staircase problem.

The motivation for examining the hump and augmented staircase problem is Lemmas 2.1 and 2.2 to follow.

Lemma 2.1 *$S(n) \leq H(n) + 1$.*

Proof: Assume Pref$([1,n]) = (\ell_1, \ldots, \ell_p)$ and Suff$([1,n]) = (r_q, \ldots, r_1)$ are known. Notice that $\ell_p = r_q$. To solve the augmented staircase problem we only have to compute the second largest element. In general it is either $x_{\ell_{p-1}}$ or $x_{r_{q-1}}$. Which one is determined by one additional comparison. The special cases where $p = 1$ or $q = 1$ are easily dealt with and require no extra comparison. $\Box$

Lemma 2.2 *For any $1 \leq h \leq k$*

$$\frac{H(h) + k - 1}{h + k - 2} n - \mathrm{O}(k) \leq \mathrm{C}(n,k) \leq \frac{H(k+1) + \lceil \log(k-1) \rceil}{k+1} n + \mathrm{O}(k).$$

Proof: We first show the upper bound for $\mathrm{C}(n,k)$.

We partition the sequence $(x_1, x_2, \ldots, x_n)$ into blocks of size $k+1$ and solve the hump problem for each block. This requires at most $\lceil \frac{n}{k+1} \rceil H(k+1)$ comparisons. Then the set-maxima for the two subsequences of length k inside each block are known. The remaining set-maxima are maxima for subsequences intersecting two consecutive blocks. Let $s_{k-1} \geq s_{k-2} \geq \cdots \geq s_1$ be the maxima of the suffixes of length $k-1, k-2, \ldots, 1$ of a block and let $p_1 \leq p_2 \leq \cdots \leq p_{k-1}$ be the maxima of the prefixes of the next block. These values are known since the hump problem has been solved for each block. We have to compute $\max\{s_{k-1}, p_1\}, \ldots, \max\{s_1, p_{k-1}\}$ in order to compute the maxima for subsequences of length k intersecting the two blocks in question. Because the two sequences are monotone the outcome will be $s_{k-1}, \ldots, s_i, p_{k+1-i}, \ldots, p_{k-1}$ for some i. We can determine i by binary search and therefore the $k-1$ maxima by $\lceil \log(k-1) \rceil$ additional comparisons for each pair of consecutive blocks. The upper bound follows.

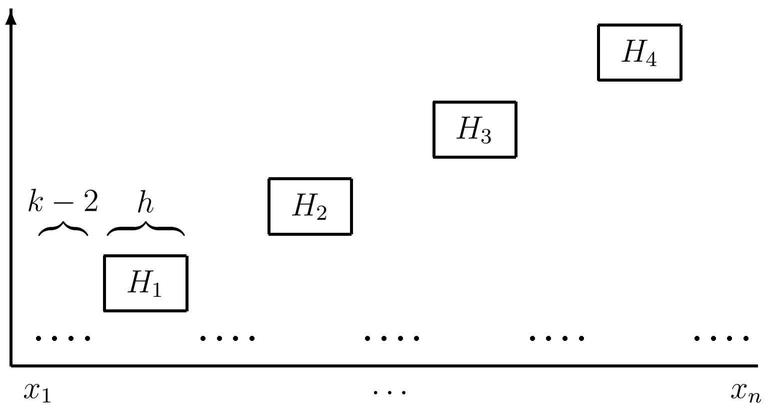

Figure 3: The ordering of the elements used by the off-line adversary strategy.

To show the lower bound we reduce a number of independent instances of the hump problem to the set-maxima problem.

The idea is that if a subsequence of length $h \leq k$ is surrounded on both sides by subsequences of length $k-2$ containing “small” elements, an algorithm solving the set-maxima problem will reveal which comparisons to make to solve the hump problem for the middle subsequence of length h.

We partition the sequence $(x_1, x_2, \ldots, x_n)$ into blocks of size $h+k-2$. The $k-2$ leftmost elements in each block are “small” (think of them as being $-\infty$). The rightmost h elements are “large”. By scaling, the “large” elements in each block are made greater than the elements in the blocks to the left of it. Figure 3 illustrates the set up.

Besides solving all the hump problems any algorithm for the set-maxima problem has to verify that all the “small” elements are indeed small. Thus the number of comparisons made is at least $(k-2)+H(h)$ per block except for the last one. We accounted for the $k-2$ and $H(h)$. In addition, comparisons have to be made between the leftmost “large” element in a block and the the last (“large”) element in the preceeding block to determine the maximum for the subsequence of length k containing the two elements referred to and the “small” elements between them. All in all, at least $H(h)+k-1$ comparisons per block except for the last one have to be made. □

The optimal choice of h to maximise $\frac{H(h)+k-1}{h+k-2}$ depends on H.

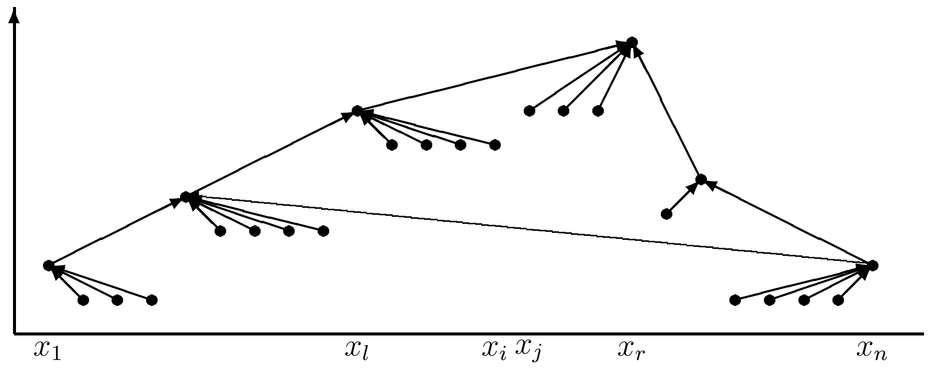

Figure 4: A snapshot of the known ordering for the hump algorithm.

We finally prove two lemmas which lead to the main theorem of the paper, namely Theorem 2.5.

Lemma 2.3 $H(n) \leq n + 2\sqrt{n} - 2$.

Proof: A simple way to solve the hump problem is to compute Pref($[1, i]$) from left to right and Suff($[j, n]$) from right to left by increasing i and decreasing j one by one and test whether a new maximum for a prefix or a suffix is met, until i and j meet ($i = j - 1$). In the extreme case only x_1 and x_n are maxima and the situation when i and j meet after $n - 2$ comparisons would be that x_1 and x_n had to be compared and the smaller of the two should be compared with up to $\frac{n}{2}$ elements (those tested smaller than the larger one). This would altogether require $\frac{3n}{2} - 1$ comparisons. We can do much better by preventing this situation to happen.

Let b be a constant to be fixed later. We augment the simple method by interrupting the propagation of i (and j) if no new maximum has been encountered for b steps. Let x_l be the last element in Pref($[1, i]$) and x_r the first element in Suff($[j, n]$) upon interruption. x_l and x_r are compared and if x_l is the smaller one, i is propagated until i and j meet or a new maximum is met. If x_r is the smaller one, j is propagated similarly. See Figure 4.

Now when i and j meet at most b comparisons have to be made. To ensure this we did at most $\lfloor \frac{n-1}{b+1} \rfloor$ interruptions and therefore at most $\lfloor \frac{n-1}{b+1} \rfloor$ comparisons. We conclude that $H(n) \leq n - 2 + b + \lfloor \frac{n-1}{b+1} \rfloor$.

If we let $b = \lfloor\sqrt{n}\rfloor - 1$ we get $H(n) \leq n + \lfloor\sqrt{n}\rfloor + \left\lfloor \frac{n-1}{\lfloor\sqrt{n}\rfloor} \right\rfloor - 3 \leq n + 2\sqrt{n} - 2$, and the lemma follows. $\square$

Lemma 2.4 $n + \left\lfloor \sqrt{2n + \frac{1}{4}} - \frac{1}{2} \right\rfloor - 3 \leq S(n)$.

Proof: The bound is proved through an adversary argument. Divide $(x_1, x_2, \ldots, x_n)$ into $b+1$ blocks of size $a, b, b-1, \ldots, 2, 1$ where $a \leq b$. Then $n = a + \frac{b(b+1)}{2}$ and $b = \left\lfloor \sqrt{2n + \frac{1}{4}} - \frac{1}{2} \right\rfloor$. The blocks are named $b+1, b, b-1, \ldots, 2, 1$. Notice that block $b+1$ might be empty.

Let ℓ_i denote the index of the leftmost element in block i for $i = b, b-1, \ldots, 2, 1$.

The adversary strategy maintains $b+1$ parameters $c_b, c_{b-1}, \ldots, c_1$ and m such that the following invariant holds true during the computation.

- For $b+1 \geq i > m$, $i > j$, all elements in block i can be made smaller than all elements in block j.
- For $m > i \geq 1$, all elements in block i can be made smaller than all elements in block m.
- For $b \geq i \geq 1$, the element x_{c_i} (the candidate in block i) can be made the maximal element within block i and if $c_i \neq \ell_i$, x_{ℓ_i} can be made the second largest element in block i.
- $c_m = \ell_m$.

The invariant is depicted in Figure 5. The parameters are initialised such that each c_i is the leftmost element in block i, i.e.

$$(c_b, c_{b-1}, \ldots, c_1; m) = (\ell_b, \ell_{b-1}, \ldots, \ell_1; 1).$$

It is quite involved to write down in detail the strategy to maintain the invariant. We describe the answers for the three cases where parameters change. For the remaining cases the answer follows from the invariant to be maintained.

When x_s and x_t $(s < t)$ from block i and j $(i \geq j)$ are compared, we call the comparison *internal* if $i = j$ and *external* otherwise. The three cases described concern all situations where x_s is an element from a block to the left of block m $(b \geq i > m)$:

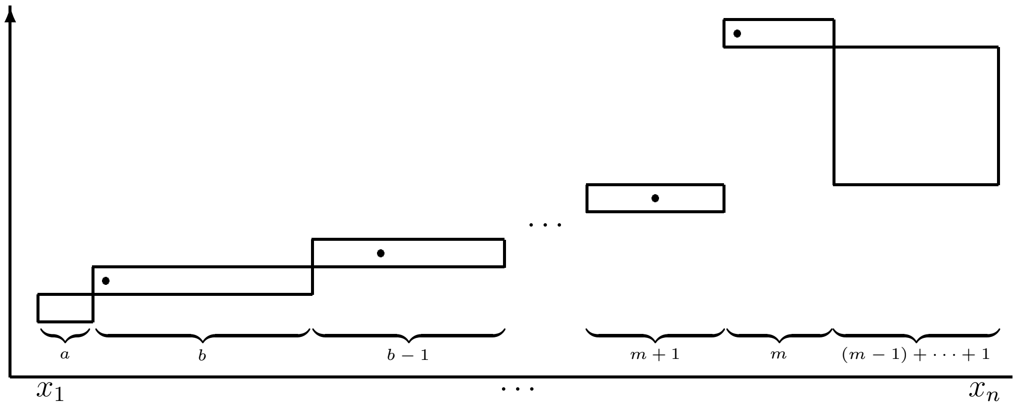

Figure 5: The ordering of elements for the augmented staircase problem. The black points indicate the x_{c_i}'s.

A. $b \geq i > m$ and the comparison is internal $(i = j)$, $c_i = \ell_i < s$ and at least one of x_s or x_t has not been compared to x_{c_i}. If x_t has not been compared to x_{c_i} then the answer given is $x_s < x_t$ and c_i is changed to t. Otherwise the answer given is $x_s > x_t$ and c_i is changed to s.

B. $b \geq i > m$ and the comparison is external $(i > j)$, $c_i = \ell_i = s$ and some element x_u in block i has not lost any internal comparison. Then the answer given is $x_s < x_t$ and c_i is changed to u.

C. $b \geq i > m$ and the comparison is external $(i > j)$, $c_i = \ell_i$ and x_{c_i} has been compared to all the other elements in the block (and won), and no elements in the block have lost an external comparison. Then m is increased to i and the answer given is $x_s > x_t$.

Let L be the set of elements having lost at least twice during a computation. Then at least $n + |L| - 1$ comparisons have been made. It is known [6] that the second largest element has to be found among those elements which have only lost to the maximal one and that all but one of those have lost twice during any computation of the second largest (and largest) element. Let $c_b, c_{b-1}, \ldots, c_1$ and m be the parameters for the invariant at termination. It follows from the invariant that L contains $\max\{0, m-2\}$ from block m since that many elements have lost to the maximal element.

In addition we will for each block to the left of block m identify an element in L. There are three cases:

1. Block i to the left of block m has been in situation A. Then the element in block i which lost when situation A occured is in L because it will eventually have lost twice.
2. Block i to the left of block m has been in situation B. Then at least two elements in block i have lost external comparisons. This is x_{ℓ_i} and x_{c_i}. Since $i-1$ internal comparisons in block i have been made, elements in block i have lost $i+1$ times in total. Consequently one of the elements must belong to L.
3. Block i to the left of block m has not been in situation A or B (or equivalently $c_i = \ell_i$). Consider the element x in block i which took part in an external comparison with an element to the right for the first time. x cannot be x_{c_i} since block i has not been in situation B or C (no block to the left of block m has been in situation C). So $x \neq x_{c_i}$ and x have lost both an internal and an external comparison. Thus x is in L.

It follows that $|L| \geq \max\{0, m-2\} + (b-m)$ and $n+b-3$ comparisons are required. □

Combining Lemmas 2.2 (with $h = k$), 2.3, and 2.4 leads to rather involved expressions for the upper and lower bound for the set-maxima problem. We choose to give a simpler result. It is not optimal, and for small values of k it is far from being optimal.

Theorem 2.5

$$\left(1+\frac{\sqrt{2k}-5}{2(k-1)}\right)n-\mathrm{O}(k) \leq \mathrm{C}(n,k) \leq \left(1+\frac{2\sqrt{k+1}+\log(k-1)}{k+1}\right)n+\mathrm{O}(k).$$

3 The on-line set-maxima problem

We define the *on-line* comparison complexity $\mathrm{C}_{\text{on-line}}(n,k)$ to be the number of comparisons needed for an on-line algorithm to solve the set-maxima problem. An on-line algorithm is required to read the sequence $(x_1, x_2, \ldots, x_n)$

from left to right and to know all set-maxima in $(x_1, x_2, \ldots, x_{i-1})$ before reading x_i.

Lemma 3.1 $\mathrm{C}_{\text{on-line}}(n,k) \leq \left\lfloor \left(2 - \frac{1}{k-1}\right) n - 1 \right\rfloor$.

P r o o f: We are going to compute $\max\{x_{i-k+1}, x_{i-k+2}, \ldots, x_i\}$ for $i = k, k+1, \ldots, n$ (in that order). We do this by computing

$$\mathrm{Suff}([1,1]), \mathrm{Suff}([1,2]), \ldots, \mathrm{Suff}([1,k-1])$$

and then $\mathrm{Suff}([i-k+1,i])$. The leftmost element in $\mathrm{Suff}([l,u])$ is the maximal element in the interval $[l,u]$. Hence all set-maxima have been computed if we compute the Suff's as described.

$\mathrm{Suff}([l,u])$ is represented by a double ended queue Q (this it what Knuth calls a *deque* [5]).

When x_i is read and we go from interval $[i-k, i-1]$ to $[i-k+1, i]$, we first remove the leftmost index of Q if it is $i-k$. Now Q represents $\mathrm{Suff}([i-k+1, i-1])$. Then we insert i into Q from the right. Before i is inserted, all indices are removed (from the right) which refer to values less than x_i. Now Q represents $\mathrm{Suff}([i-k+1,i])$ and the leftmost index refers to $\max\{x_{i-k+1}, x_{i-k+2}, \ldots, x_i\}$ which is the new output.

To count the number of comparisons involved we focus on how many comparisons any x_i can lose. It might lose one comparison when i is inserted into Q and again when i is removed. This immediately gives an upper bound $2n$.

During a computation Q represents interchangeably intervals of length k and $k-1$. If we view a step as going from one interval $[i-k+1, i-1]$ of length $k-1$ to the next interval $[i-k+2, i]$ of length $k-1$ by first inserting i from the right and then removing the leftmost index from Q if it equals $i-k+1$. We can observe that each time the leftmost index in Q changes, it is either because x_i does not lose any comparisons during the insertion of i and i becomes the new leftmost index or it is because the index $i-k+1$ is removed without x_{i-k+1} losing a comparison. In both cases one comparison is saved. Since Q cannot have the same leftmost index for k consecutive intervals of length $k-1$, at least $\lceil \frac{n}{k-1} \rceil - 1$ comparisons are saved. Also x_1 is inserted without losing and x_n is never removed. So

$$\mathrm{C}_{\text{on-line}}(n,k) \leq 2n - 2 - \left(\lceil \tfrac{n}{k-1} \rceil - 1\right)$$

and the lemma follows. □

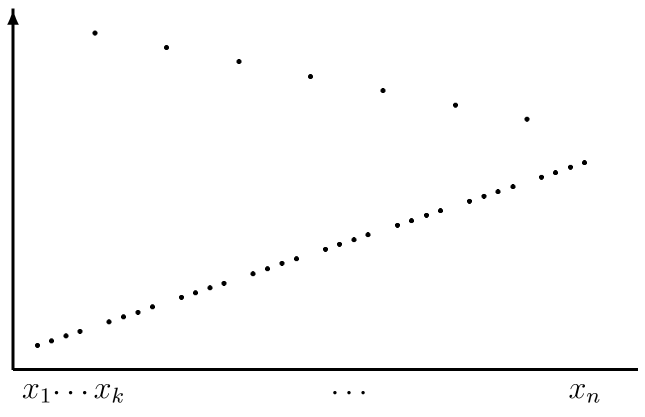

Figure 6: The ordering enforced by the adversary strategy.

Lemma 3.2 $\left(2 - \frac{1}{k-1}\right) n - \mathrm{O}(k) \leq \mathrm{C}_{\text{on-line}}(n, k)$.

Proof: We use an adversary strategy which on $x_i?x_j$ for $i < j$ answers:

$$x_i > x_j \text{ if } k-1 \text{ divides } i, \text{ and } x_i < x_j \text{ otherwise.}$$

The ordering that we enforce on $(x_1, x_2, \ldots, x_n)$ can be seen in Figure 6.
Let $1 < p$ and $(p+2)(k-1) \leq n$. We argue that $k-1$ elements

$$x_{p(k-1)+1}, x_{p(k-1)+2}, \ldots, x_{(p+1)(k-1)}$$

will lose at least $2k-3$ comparisons altogether. Having argued that, the lemma follows.
Since $x_{p(k-1)} = \max\{x_{(p-1)(k-1)+i}, \ldots, x_{p(k-1)+i}\}$ for $i = 1, 2, \ldots, k-1$ any on-line algorithm has to compare $x_{p(k-1)+i}$ to $x_{p(k-1)}$ where $x_{p(k-1)+i}$ loses.
Since $x_{(p+1)(k-1)} = \max\{x_{p(k-1)+1}, \ldots, x_{(p+1)(k-1)+1}\}$ it has to be verified that $x_{p(k-1)+i} < x_{(p+1)(k-1)}$ for $i = 1, 2, \ldots, k-2$. This *cannot* follow by transitivity from the comparisons involving $x_{p(k-1)}$ above. Thus $x_{p(k-1)+i}$ has to lose at least once more (for $1 \leq i < k-1$) and the statement follows. □

The following theorem is a direct corollary of Lemmas 3.1 and 3.2.

Theorem 3.3 $\mathrm{C}_{\text{on-line}}(n, k) = \left(2 - \frac{1}{k-1}\right) n - \mathrm{O}(k)$.

References

[1] A. E. Andreev. On a family of boolean matrices. *Moscow Univ. Math. Bull.*, 41(2):97–100, 1986.

[2] Amotz Bar-Noy, Rajeev Motwani, and Joseph Naor. A linear time approach to the set maxima problem. *SIAM Journal on Discrete Mathematics*, 5(1):1–9, 1992.

[3] James Cooper and Leslie Kitchen. CASOP: A fast algorithm for computing the n-ary composition of a binary associative operator. *Information Processing Letters*, 34:209–213, 1990.

[4] Wayne Goddard, Claire Kenyon, Valerie King, and Leonard J. Schulman. Optimal randomized algorithms for local sorting and set-maxima. *SIAM Journal of Computing*, 22(2):272–283, 1993.

[5] Donald E. Knuth. *The Art of Computer Programming, Volume I: Fundamental Algorithms.* Addison-Wesley, Reading, MA, 1968.

[6] Donald E. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching.* Addison-Wesley, Reading, MA, 1973.

[7] Kurt Mehlhorn. Some remarks on boolean sums. *ACTA Informatica*, 12:371–375, 1979.

[8] Ingo Wegener. Boolean functions whose monotone complexity is of size $n^2/\log n$. *Theoretical Computer Science*, 21:213–224, 1982.