A note on self-improving sorting with hidden partitions

Siu-Wing Cheng†, Man-Kwun Chiu‡, Kai Jin†
† HKUST, Hong Kong. ‡ Freie University at Berlin, Germany

1 Introduction.

The sorting problem under a so-called “self-improving computational model” was studied in [1]: In this model, we will have input instances $I_1, I_2, \ldots, \text{etc}$ generated as follows. An instance $I$ contains $n$ elements $x_1, \ldots, x_n$, and its $i$-th ($1 \leq i \leq n$) element $x_i^I$ is generated according to a distribution $D_i$. The $n$ distributions $D_1, \ldots, D_n$ are fixed but are not given. The target is to compute and output $\pi(I)$ – the ranks of the $n$ elements in $I$.

Let $H(\pi(I))$ denote the entropy of the output $\pi(I)$. The authors in [1] showed that they can design a learning phase which learns the distributions and builds some data structures by analyzing several instances so that for a given $I$ in the operation phase, they can compute $\pi(I)$ in $O(H(\pi(I)) + n)$ expected time, which matches the information theory lower bound.

We study in this paper a more general setting which allows some dependency among the $n$ elements. We assume that the $n$ elements are partitioned into $g$ groups (each element belongs to exactly one group) and in the $k$-th ($1 \leq k \leq g$) group there is a variable $z_k$ which is generated according to a fixed distribution $D_k$ and each element in this group is a function of $z_k$. Note that the partition as well as the $g$ distributions $D_1, \ldots, D_g$ are not given.

However, we need to impose some constraints on these functions of $z_k$. Assume that the $k$-th group contains $n_k$ elements $x_1, \ldots, x_{n_k}$ and moreover $x_1 = f_1(z_k), \ldots, x_{n_k} = f_{n_k}(z_k)$.

We assume that each function $f_i()$ can have at most $\mu$ extremal points and every pair of functions $f_i()$ and $f_j()$ can have at most $\sigma$ intersections, where $\mu$ and $\sigma$ are known constants.

Under such constraints, our result is the following.

\begin{itemize}
  \item [\textbf{Theorem 1.}] In operation phase, we can compute $\pi(I)$ in $O(H(\pi(I)) + n)$ expected time.
\end{itemize}

1.1 Technique overview.

Learning phase overview. We learn the hidden partition using constant many instances. Also, we construct the V-list in the same way as in [1]. Precisely, take $\lambda = \lceil \log n \rceil$ instances and merge all the $\lambda \cdot n$ elements in these instances into a big list and sort them in increasing order; denote the results by $y_1, \ldots, y_{\lambda n}$. Assign $V_r = y_r \cdot \lambda (1 \leq r \leq n)$, $V_0 = -\infty$, and $V_{n+1} = +\infty$. We call $V_r$ the predecessor of $x_i$ if $x_i \in [V_r, V_{r+1})$. For the $k$-th ($1 \leq k \leq g$) group, the predecessors of the $n_k$ elements in this group respectively and the order between these elements are denote by $p_{0_k}$; its entropy denoted by $H(p_{0_k})$. Finally, let $n' = \max_k n_k$, and we sample $T = n'(n(\mu + 1) + n'\sigma) \log n$ instances to learn the distribution of $p_{0_k}$.

Operation phase. First, we compute $p_{0_k}$ for each $k$ ($1 \leq k \leq g$). Second, for each $k$, denote $\sigma_k$ the list of $n_k$ elements in $k$-th group in sorted order, find all $r$ such that $\sigma_k \cap [V_r, V_{r+1})$ is nonempty, and put the sublist $\sigma_k \cap [V_r, V_{r+1})$ into $S_r$ (So $S_r$ is a set of sublists). Third, we use a concatenation to merge all the sublists in $S_r$ into one list $s_r$ in sorted order. Finally, by concatenating $s_0, \ldots, s_n$, we obtain the sorted list of all elements.

1.2 Running time analysis of the operation phase.

We need the following three crucial lemmas.
A note on self-improving sorting with hidden partitions

Lemma 2. For each $k$ ($1 \leq k \leq g$), we can compute $po_k$ in $O(H(po_k) + n_k)$ time.

Lemma 3. $\sum_k H(po_k) = H(\pi(I)) + O(n)$.

Lemma 4. With high probability, on our construction of the $V$-list, it is guaranteed that for each $r$, the expected size of $S_r$ (i.e., the number of sublists in $S_r$) is a constant.

By Lemma 2 the first step runs in $O(\sum_k H(po_k) + n_k)$ time, which is $O(\sum_k H(po_k)) + O(n) = H(\pi(I)) + O(n)$ time further according to Lemma 3. The second and last step cost $O(n)$ time. The third step takes $O(n)$ time by applying Lemma 4. Thus we get Theorem 1.

Lemma 4 follows from Lemma 2.3 of [1] because we can compute $(po_1, \ldots, po_g)$ in $O(n)$ comparisons given $\pi(I)$. Lemma 4 is the same as Lemma 6 in [2]. Lemma 2 is proved below.

2 Learning phase I – compute the hidden partition in $\mu^4$ rounds

Assume we want to determine whether $(x_1, x_2)$ is in the same group.

Recall that each function has at most $\mu$ extremal points. We take $m = \mu^4$ samples of $(x_1, x_2)$. Denote the values by $(x_{11}, x_{21}), \ldots, (x_{1m}, x_{2m})$. Without loss of generality, assume that $x_{11} \leq x_{1,2} \leq \ldots \leq x_{1,m}$. (Otherwise we make it so by sorting)

Moreover, for any sequence of numbers $(A_1, \ldots, A_m)$ with length $m$, we define function $D(A_1, \ldots, A_m)$ as the minimum number $d$ such that $(A_1, \ldots, A_m)$ can be partitioned into $d$ monotonic sub-sequence. A sub-sequence is monotonic if it is either increasing or decreasing.

We can prove that

- If $x_1$ and $x_2$ are in the same group, $D(x_{11}, \ldots, x_{2,m}) \leq 2\mu + 1$;
- If $x_1$ and $x_2$ are in different groups, $D(x_{11}, \ldots, x_{2,m}) = \Omega(\mu^4)$.

Therefore,

- If $D(x_{11}, \ldots, x_{2,m}) \leq 2\mu + 1$, with high probability $(x_1, x_2)$ are in the same group.
- If $D(x_{11}, \ldots, x_{2,m}) > 2\mu + 1$, it is definitely true that $(x_1, x_2)$ are in different groups.

As a consequence, we can learn the hidden partition easily by calling function $D$.

Moreover, since $\mu$ is a constant, so as $m$, hence it only costs constant time to compute $D$.

3 Learning phase II – learn the distribution of $po_k$

We need to introduce some notation here.

For convenience, assume that $x_1, \ldots, x_{n_k}$ are in the $k$-th group.

![Figure 1](image)

**Figure 1** Illustration of the arrangement.

First, we draw $n_k$ curves $y = f_1(z), \ldots, y = f_{n_k}(z)$. Moreover, for each $r$ ($1 \leq r \leq n$), we draw a horizontal line $y = V_r$. Let $A$ denote the arrangement of these $n + n_k$ curves.

For each intersection in $A$, we draw a vertical line, as shown in Figure 1. According to our assumption on the functions, there are less than $W = n_k n (\mu + 1) + n_k^2 \sigma$ such intersections.
These intersections divide the plane into at most $W$ slabs. Notice that $p_{0_k}$ remains the same when $z_k$ is restricted to any fixed slab, yet it could be the same for different slabs. Thus there are at most $W$ possible (different) choices of $p_{0_k}$, denoted by $r_1, \ldots, r_W$. Moreover, let $p_i$ be the probability that $p_{0_k}$ is identical to $r_i$. Note that $W^*, p_i, r_i$ are all unknown and we do not build $A$ explicitly. Remind that the entropy $H(p_{0_k})$ is simply defined as $\sum_i p_i \log(1/p_i)$.

In learning phase, we take $T \geq W \log n$ instances to sample the results of $p_{0_k}$ and count their frequency. For $1 \leq i \leq W^*$, denote by $\chi_i$ the times that $r_i$ is sampled. Let $q_i = \chi_i/T$. (Note that $\chi_i$ might be zero for some $r_i$; such $r_i$ is unknown to us. Other $r_i$'s are known.)

### 3.1 Store all the sampled results of $p_{0_k}$ in a trie

We encode every known result of $p_{0_k}$ by a vector $(b_1, \ldots, b_{n_k})$ (similar to the Lehmer code).

**Definition 5.** Given a known result of $p_{0_k}$, element $b_1$ is defined as among $V_0, \ldots, V_n$ the predecessor of $x_1$; and $b_2$ is defined as among $V_0, \ldots, V_n, x_1$ the predecessor of $x_2$; so on and so forth; finally, $b_{n_k}$ is defined as the predecessor of $x_{n_k}$ among $V_0, \ldots, V_n, x_1, \ldots, x_{n_k-1}$.

Four examples are given in Figure 2 (a). The bottom of the columns shows the vectors.

![Figure 2](Illustration of the encoding given in Definition 5 and the trie.)

We store the vectors of all sampled results of $p_{0_k}$ into a trie as shown in Figure 2 (b). Moreover, we assign every node in this trie a weight: A leaf labeled by $r_i$ has weight $q_i$, and the weight of an internal node equals the total weight of its sons; so the root has weight 1.

### 4 Operation phase Step 1 – compute $p_{0_k}$

First, let us consider an ideal case where $q \equiv p$, i.e. $q_i = p_i$ for every $1 \leq i \leq W^*$.

Assume we are given the values of $(x_1, \ldots, x_{n_k})$ and we want to determine $p_{0_k}$. Equivalently, we want to determine the vector corresponding to $p_{0_k}$. Similar as what Fredman did in [3], using $(x_1, \ldots, x_{n_k})$, we can compute $b_1, \ldots, b_{n_k}$ step by step. When $p_{0_k} = r_i$, this process corresponds to a path in the trie starting from the root to the leaf labeled with $r_i$.

According to some basic algorithmic knowledge (see section 3.2 paragraph 1 in [1]), if currently we are at a node with weight $w_j$ and the next round we proceed to a son with weight $w_k$, the time for choosing the son in this step would be $O(1 + \log(w_j/w_k))$. Therefore, if $p_{0_k} = r_i$, it takes $O(n_k + \log(1/q_i))$ time to reach the node labeled with $r_i$.

Further since the probability that “$p_{0_k} = r_i$” is $p_i$, the expected time for computing $p_{0_k}$ would be $O(\sum_i p_i(n_k + \log(1/q_i))) = O(n_k + \sum_i p_i \log(1/q_i)) = O(n_k + H(p_{0_k}))$ when $q \equiv p$.

Next, we show that even if $q \not\equiv p$, the expected running time is still $O(n_k + H(p_{0_k}))$. 

4.1 The proof of Lemma 2

Denote \( q = (q_1, \ldots, q_W) \). Let \( t_i^q \) be the time for computing \( p_0^k \) when \( p_0^k = r_i \) and when our sampling result is some fixed \( q \). Similar as in the above case, for \( q_i > 0, \) we compute \( p_0^k \) in time \( O(n_k + \log(1/q_i)) \) when \( p_0^k = r_i; \) yet for \( q_i = 0, \) we find no result after searching the trie and we use a trivial method to compute \( p_0^k \) and it costs \( O(n_k \cdot \log n) \) time. Therefore,

\[
t_i^q = \begin{cases} 
O(n_k + \log(1/q_i)), & q_i > 0; \\
O(n_k \cdot \log n), & q_i = 0.
\end{cases}
\]

(1)

Thus the expected running time for computing \( p_0^k \) in operation phase is given by

\[
\sum_q \Pr(q) \cdot \sum_t t_i^q = \sum_{i} p_i \sum_q \Pr(q) t_i^q
= \sum_i p_i \sum_{q,q_i > 0} \Pr(q) O(n_k + \log(1/q_i)) + \sum_i p_i \sum_{q,q_i = 0} \Pr(q) O(n_k \log n)
\]

(2)

The second term is \( O(n_k \log n \sum_i p_i (1 - p_i)^T) \leq O(n_k \log n W^*/(T + 1)) = O(n_k) \). (3)

The first term is \( \sum_i p_i \sum_{q,q_i > 0} \Pr(q) O(n_k) + \sum_i p_i \sum_{q,q_i > 0} \Pr(q) O(\log(1/q_i)) \)

\[
\leq O(n_k) + \sum_i p_i \sum_{j=1}^T \Pr(q_i = j/T) O(\log(T/j))
\]

(4)

\[
= \sum_i p_i \sum_{1 \leq j \leq T/2} \Pr(q_i = j/T) O(\log(T/j)) + \sum_i p_i \sum_{p_i T/2 < j \leq T} \Pr(q_i = j/T) O(\log(T/j))
\]

(5)

\[
\leq \sum_i p_i \sum_{1 \leq j \leq T/2} \Pr(q_i = j/T) O(\log T) + \sum_i p_i \sum_{p_i T/2 < j \leq T} \Pr(q_i = j/T) O(\log(2/p_i))
\]

(6)

The second term \( \leq \sum_i p_i O(\log(2/p_i)) = O(1 + H(p_0^k)) \).

To bound the first term, we need to bound \( \sum_{1 \leq j \leq T/2} \Pr(q_i = j/T) < \Pr(q_i \leq p_i/2) \), for which we apply the Chernoff bound. Note that the expectation of \( q_i \) is given by \( p_i \), so \( \Pr(q_i \leq p_i/2) \leq e^{-p_i T/8} \leq \frac{\log T}{8 T} \). Hence the first term \( \leq \sum_i p_i \frac{\log T}{8 T} = O(W' \log T/T) = O(1) \).

To sum up, altogether we prove that the expected running time is \( O(n_k + H(p_0^k)) \).

References

1. N. Ailon, B. Chazelle, K. Clarkson, D. Liu, W. Mulzer, and C. Seshadhri. Self-improving algorithms. *SIAM Journal on Computing*, 40(2):350–375, 2011. [doi:10.1137/090766437]

2. S. Cheng and L. Yan. Extensions of self-improving sorters. In 29th *International Symposium on Algorithms and Computation*, ISAAC 2018, December 16-19, 2018, Jiaoxi, Yilan, Taiwan, pages 63:1–63:12, 2018. [doi:10.4230/LIPIcs.ISAAC.2018.63]

3. M.L. Fredman. How good is the information theory bound in sorting? *Theoretical Computer Science*, 1(4):355–361, 1976. [doi:https://doi.org/10.1016/0304-3975(76)90078-5]