2005 International Conference on Analysis of Algorithms DMTCS proc. **AD**, 2005, 353–356

Distribution-sensitive set multi-partitioning

Amr Elmasry

Computer Science Department
Alexandria University, Egypt
elmasry@alexeng.edu.eg

Given a set $\mathcal{S}$ with real-valued members, associated with each member one of two possible types; a multi-partitioning of $\mathcal{S}$ is a sequence of the members of $\mathcal{S}$ such that if $x, y \in \mathcal{S}$ have different types and $x < y$, x precedes y in the multi-partitioning of $\mathcal{S}$. We give two distribution-sensitive algorithms for the set multi-partitioning problem and a matching lower bound in the algebraic decision-tree model. One of the two algorithms can be made stable and can be implemented in place. We also give an output-sensitive algorithm for the problem.

Keywords: algorithm analysis and design; distribution-sensitive algorithms; output-sensitive algorithms; lower bounds

1 Introduction

An output-sensitive algorithm is an algorithm for which the running time relies on the size of its output in addition to its input size. For example, Kirkpatrick and Seidel [6] have shown that the complexity of constructing the convex hull of a set of n points is $\Theta(n \log h)$, where h is the number of hull vertices.

Alternatively, a distribution-sensitive algorithm is an algorithm whose running time relies on how the distribution of the input affects the output. For example, Munro and Spira [11] have shown that the complexity of sorting a multiset of n elements is $\Theta(n \log n - \sum_i n_i \log n_i + n)$, where n_i is the number of elements with the ith largest value. Sen and Gupta [12] gave distribution-sensitive algorithms for some geometric problems including the convex hull.

To see that the latter paradigm is superior to the former, we note that $n \log n - \sum_i n_i \log n_i$ is maximized at $n_i = n/h$, where h is the number of distinct n_i's, implying an $O(n \log h)$ bound for sorting multisets. Furthermore, in many cases, especially for very non-uniform distribution of the n_i's, $\Theta(n \log n - \sum_i n_i \log n_i)$ can be much less than $\Theta(n \log h)$. Consider the case where all n_i's equal 1 except n_h, and $h \leq n/\log n$. In such case $n \log n - \sum_i n_i \log n_i$ is linear, in contrast with $n \log h$ which is $O(n \log n)$.

Consider a set $\mathcal{S}$ of n real-valued members, associated with each member one of two possible types. Let $\mathcal{S}'$ be the set of n' members of one type, and $\mathcal{S}''$ be the set of n'' members of the other type $(n = n' + n'')$. We define a multi-partitioning of $\mathcal{S}$ to be a sequence of the members of $\mathcal{S}$ such that if $x, y \in \mathcal{S}$ have different types and $x < y$, x precedes y in this sequence; this results in partitioning $\mathcal{S}$ into alternating blocks of maximal subsequences of members with the same type. In other words, for any two consecutive blocks, the members of one block are of one type, and the members of the other block are of the other type. Without loss of generality, we assume that if members of different types have the same value, the members of $\mathcal{S}'$ appear before those of $\mathcal{S}''$.

In the next two sections, we give two distribution-sensitive algorithms for set multi-partitioning that run in $O(n \log n - \sum_i n_i \log n_i + n)$, where n_i is the count of the members of the ith block. In Section 4, we give a matching lower bound in the algebraic decision-tree model. In Section 5, we give an output-sensitive algorithm that runs in $O(n \log h)$, where h is the number of blocks.

Assume that the members of $\mathcal{S}$ are sorted according to the corresponding values; this results in a multi-partitioning of $\mathcal{S}$ into the required blocks. The basic idea of our algorithms is to produce such blocks in sequence, without ordering the members within individual blocks.

2 A randomized divide-and-conquer algorithm

1. Assume, without loss of generality, that $n' \geq n''$.
2. Pick a member uniformly at random from those of $\mathcal{S}'$, and call it x.

3. Partition the members of $\mathcal{S}''$ into $\mathcal{S}''_L$ and $\mathcal{S}''_R$. The values of the members of $\mathcal{S}''_L$ are smaller than that of x, while those of $\mathcal{S}''_R$ are larger than the value of x. Let y be the member with the largest value in $\mathcal{S}''_L$ and z be that with the smallest value in $\mathcal{S}''_R$.

4. Partition the members of $\mathcal{S}'$ into $\mathcal{S}'_L$, $\mathcal{S}'_M$ and $\mathcal{S}'_R$. The values of the members of $\mathcal{S}'_L$ are smaller than that of y, the values of the members of $\mathcal{S}'_M$ are larger than that of y and smaller than that of z, while those of $\mathcal{S}'_R$ are larger than that of z.

5. Apply the algorithm recursively for the subsets $\mathcal{S}'_L$ and $\mathcal{S}''_L$, and output the corresponding blocks.

6. Follow these blocks by the block $\mathcal{S}'_M$.

7. Apply the algorithm recursively for the subsets $\mathcal{S}'_R$ and $\mathcal{S}''_R$, and output the corresponding blocks.

Indeed, the fact that the above algorithm correctly identifies the ordering of the blocks follows by noting that the values of $\mathcal{S}'_M$ are larger than those of $\mathcal{S}'_L$ and $\mathcal{S}''_L$, and smaller than those of $\mathcal{S}'_R$ and $\mathcal{S}''_R$.

Theorem 1 *The expected running time of the above algorithm is* $O(n \log n - \sum_i n_i \log n_i + n)$.

Proof. We say that the element x results in a good split if the rank of x within the sorted sequence of $\mathcal{S}'$ turns out to be between $n'/4$ and $3n'/4$. Hence, a good split, resulting from randomly selecting x, happens with probability $1/2$.

A crucial observation for the time bound analysis is that x belongs to $\mathcal{S}'_M$. For a good split, the number of members of $\mathcal{S}'_L$ is less than $3n'/4$ (i.e. $|\mathcal{S}'_L| < 3n'/4$). It follows that $|\mathcal{S}'_L| + |\mathcal{S}''_L| < 3n'/4 + n'' = n - n'/4$. Since $n' \geq n/2$, $|\mathcal{S}'_L| + |\mathcal{S}''_L| < 7n/8$. Symmetrically, $|\mathcal{S}'_R| + |\mathcal{S}''_R| < 7n/8$.

The time spent in all the steps of the algorithm other than the recursive calls is linear. We charge a constant cost for every member of the two subsets involved in each recursive call. Consider a member e of a block of length n_i of the resulting partitioning. Let C_e be the random variable representing the number of recursive calls which involve e. This random variable obeys a negative binomial distribution, with the success represented by a call resulting in a good split. Since the size of each of the two subproblems is reduced by at least a factor of $7/8$ with each good split, the number of good splits that e contributes to is at most $\log_{8/7}(n/n_i) + O(1)$. Hence, the expected value for C_e is at most $2\log_{8/7}(n/n_i) + O(1)$. It follows that the expected running time of the algorithm is $O(\sum_i n_i \log(n/n_i) + n) = O(n \log n - \sum_i n_i \log n_i + n)$. □

The above algorithm can be easily derandomized by choosing x at step 2 of the algorithm to be the median of $\mathcal{S}'$ instead of picking it at random. This median-finding can be done in linear time, which will not affect the overall bound. (For this deterministic version, we have $|\mathcal{S}'_L| + |\mathcal{S}''_L| < 3n/4$ and $|\mathcal{S}'_R| + |\mathcal{S}''_R| < 3n/4$, resulting in the members of the ith block contributing to at most $\log_{4/3}(n/n_i) + O(1)$ recursive calls.)

The algorithm can be implemented to run in place, only with a constant extra space, as follows. We use the in-place median-finding of Lai and Wood [7] (in case we choose to implement the deterministic version), and the standard in-place partitioning as in the Quicksort [5] algorithm. To avoid the recursion stack, the same trick, based on stoppers, used by Ďurian [4] to implement Quicksort in place can be used. After every partitioning phase we must employ an in-place transposition of the elements. The algorithm can be further made stable (the order of the elements of a block is maintained as that of the input arrays) while implemented in place, as follows. We use the in-place stable median-finding of Katajainen and Pasanen [8], and their in-place stable partitioning [9] (here they allow integer packing).

3 A heap-based algorithm

Munro and Raman [10] gave a simple heap-based multiset sorting algorithm. We adapt their algorithm for our set multi-partitioning problem. The first step is to build a heap on the members of the the given set $\mathcal{S}$, ordered by their corresponding values. It follows that all the elements of the first block of the multi-partitioning of $\mathcal{S}$ form a sub-tree rooted at the top of the heap. Let n_i be the count of the members of this block. This makes it easy to locate these elements in $O(n_i)$, using a depth-first search from the root. Every time a member of the other type is encountered the search retreats to the parent node. Instead of discarding the root as in normal Heapsort, we start deleting the n_i elements in decreasing distance from the root. Recall that if an element is at distance d from the bottom of the heap, then deletion and the subsequent restoration of the heap takes $O(d)$ time. Munro and Raman show that the maximum number of comparisons are required when the n_i elements form a balanced tree, which yields a running time of $O(n_i \log (n/n_i) + n_i)$ for this process. We repeat the same process by deleting the elements of each of the next blocks, block after block, in sequence. The running time of Theorem 2 follows.

Theorem 2 *The total running time of the heap-based algorithm is* $O(n \log n - \sum_i n_i \log n_i + n)$.

4 The lower bound

We use the following general result of Ben-Or [1].

Ben-Or's Theorem Let W be a subset of $\mathbb{R}^d$ and let $\#W$ represent the number of disjoint connected components of W. Then, any algebraic decision tree that correctly determines the membership query $x \in W$ has $\Omega(\log \#W - d)$ height.

Consider the following decision problem, which is linear-time reducible to our set multi-partitioning problem: Given two sets $\mathcal{S}'$ of n' members, and $\mathcal{S}''$ of n'' members, and a distribution vector $(n_1, n_2, \dots)$. Determine whether this distribution vector resembles the sizes of the blocks of the corresponding set multi-partitioning problem.

Each vector of the n real values of the input sets can be considered a point in $\mathbb{R}^n$. Consider the number of ways of assigning n' members of the first type and n'' members of the second type to consecutive alternating blocks whose sizes have the distribution vector $(n_1, n_2, \dots)$. All such assignments correspond to YES instances. We show next that each such assignment corresponds to a disjoint component in $\mathbb{R}^n$.

Consider any two distinct assignments corresponding to YES instances. Pick a point in $\mathbb{R}^n$ for each of these instances, and call the two points p_1 and p_2. Then, there are at least two indices of p_1 and p_2 whose corresponding members appear in different blocks. Let the kth block be one of these blocks. Hence, any continuous path between p_1 and p_2 in $\mathbb{R}^n$ must necessarily pass through a point that corresponds to an input vector resulting in the kth block with count $\neq n_k$, which is a NO instance.

The number of such assignments corresponding to disjoint components is $\frac{n'!n''!}{n_1!n_2!\dots}$. Hence, $\#W > \frac{n/2!}{n_1!n_2!\dots}$. By simplifying using Stirling's approximation and applying Ben-Or's theorem together with the natural $\Omega(n)$ lower bound, the following theorem follows.

Theorem 3 *Any algebraic computation tree for the set multi-partitioning problem with distribution vector* $(n_1, n_2, \dots)$ *has height* $\Omega(n \log n - \sum_i n_i \log n_i + n)$.

5 An output-sensitive algorithm

1. Split S' and S'' into $\lceil n'/m \rceil$ and $\lceil n''/m \rceil$ arbitrary groups each of size at most m (m is a parameter that will be determined later).
2. Sort each of these groups using any $O(m \log m)$ sorting algorithm.
3. Repeat at most m times alternating between the two sets S' and S''
   - 3.1. Find the member with the smallest value among the groups of one of the two sets.
   - 3.2. Use binary search to find and remove all the members with a smaller value within each of the groups of the other set.

Lemma 1 *The above procedure is designed to run in* $O(n \log m)$.

Proof. There are $O(n/m)$ groups altogether. The sorting of step 2 takes $O(m \log m)$ per group, for a total of $O(n \log m)$. Finding the smallest member at step 3.1 requires checking the minimum of each of $O(n/m)$ groups. This takes $O(n)$ time for all the m iterations. The binary search at step 3.2 requires $O(\log m)$ per group, for a total of $O((n/m) \log m)$ per iteration. This takes $O(n \log m)$ for all the m iterations. □

Let h be the number of blocks of the required multi-partitioning of $\mathcal{S}$. If we set m to h, this procedure partitions the given set correctly in $O(n \log h)$. This follows from the fact that every iteration of step 3 correctly produces one of the blocks of the required multi-partitioning in the correct order.

Unfortunately, we do not know in advance the value of h. To overcome this problem, we use a standard technique that is used, for example, in [2, 3]. We call the above procedure repeatedly with $m = 2^{2^i}$ ($i = 1, 2, \dots$), every time checking if the procedure terminates after partitioning the input sequence or not. This check can be easily done, for example, by counting the number of elements of the blocks produced so far. We stop the execution once the partitioning is completed.

Theorem 4 *The running time of the above algorithm is* $O(n \log h)$

Proof. Using Lemma 1, the running time of the ith call to the above procedure is $O(n \log 2^{2^i}) = O(n2^i)$. It follows that the running time of the algorithm is

$$O(\sum_{i=1}^{\lceil \log \log h \rceil} n2^i) = O(n \log h).$$

□

References

[1] M. Ben-Or. *Lower bounds for algebraic computation trees.* 15th ACM STOC (1983), 80-86.

[2] T. Chan, *Output-sensitive results on convex hulls, extreme points, and related problems.* Discrete and Computational Geometry 16 (1996), 369-387.

[3] B. Chazelle and J. Matoušek, *Derandomizing an output-sensitive convex hull algorithm in three dimensions.* Computational Geometry: Theory and Applications 5 (1995/96), 27-32.

[4] B. Ďurian. *Quicksort without a stack.* 12th MFCS, LNCS 233 (1986), 283-289.

[5] C. Hoare. *Quicksort.* Computer Journal 5(1) (1962), 10-15.

[6] D. Kirkpatrick and R. Seidel. *The ultimate planar convex hull algorithm.* SIAM J. Comput. 15(1) (1986), 287-299.

[7] T. Lai and D. Wood. *Implicit selection.* 1st SWAT, LNCS 318 (1988), 14-23.

[8] J. Katajainen and T. Pasanen. *Stable minimum space partitioning in linear time.* BIT 32 (1992), 580-585.

[9] J. Katajainen and T. Pasanen. *Sorting multiset stably in minimum space.* Acta Informatica 31 (1994), 410-421.

[10] J.I. Munro and V. Raman. *Sorting multisets and vectors in-place.* 2nd WADS, LNCS 519 (1991), 473-479.

[11] J.I. Munro and P. Spira. *Sorting and searching in multisets.* SIAM J. Comput. 5(1) (1976), 1-8.

[12] S. Sen and N. Gupta. *Distribution-sensitive algorithms.* Nordic J. Comput. 6(2) (1999), 194-211.