The CSI Journal on
Computer Science and Engineering
Vol. 15, No. 2, 2018
Pages 1-4
Regular Paper

Progressive sorting in the external memory model

Amir Mesrikhani[1] Mohammad Farshi[1]

[1]Combinatorial and Geometric Algorithms Lab, Department of Mathematical Sciences, Yazd University, Yazd, Iran.

Abstract

Executing all steps of an algorithm that faces with massive input data may take a long time. A progressive algorithm solves the problem step by step, and produces a partial solution in each step which approximates the final solution. Therefore, the user can decide to stop the algorithm or continue to get better solutions. In this paper, we consider the problem of sorting a set of N real numbers, and design a progressive algorithm with $O(\log_{M/B} N/B)$ steps that takes $O(N/B)$ I/O operations in each step in the external memory model. The upper bound for the error of the partial solution in step r is $O(\frac{N}{(M/B)^{r/2}})$.

Keywords: Progressive algorithm, Partial solution, Sorting problem, External memory algorithm.

1. Introduction

The massive data raise two important challenges in designing algorithms. First, consider an algorithm that processes a large amount of input data. Executing all steps of the algorithm to compute the final output may take a long time. Therefore, it is interesting to design an algorithm that generates partial solutions in some specific steps. The partial solution approximates the final solutions with respect to an error function. Second, massive data do not fit into the main memory. Therefore, the data is maintained in the secondary memory like a disk. We know that the time of transferring some blocks of data between the main memory and the secondary memory usually overcomes the time of processing data in the processor. Consequently, for processing massive data, one has to develop an algorithm such that it is efficient according to the memory hierarchy. These issues motivate us to design algorithms that solve problems with a large amount of input data step by step. Also, these algorithms should be efficient in term of the transferred blocks of data between the main memory and the secondary memory.

1.1. Progressive algorithm

Informally, an algorithm that solves a problem step by step and reports a partial solution with guaranteed error bound in each step is called a *progressive algorithm* [1]. Reporting the partial solutions to the user is the main idea of the progressive algorithms. The user of these algorithms, based on the error of each partial solution, can stop the running of the algorithm or decide to continue to produce better partial solutions whose error is less than the error of the previous partial solutions. Let g_r be the partial solution in step r. The error of g_r is measured by an error function. This function takes g_r as an argument and outputs a nonnegative value as the error value of g_r. The speed of convergence of the partial solution to the final solution is computed by a convergence function. This function takes the step number and outputs the upper bound of the error of the corresponding partial solution. In 2014, Alewijnse et al. [1] proposed the following formal definition for the progressive algorithm.

Definition 1. A k-step progressive algorithm with respect to the error function f_e and the convergence function *conv*, is an algorithm that generates the partial solution g_r in step r such

that:

$$f_e(g_r) \leq conv(r)$$

1.2. The External memory model

The Main assumption in the Random Access Memory (RAM) model is the infinite size of the main memory. So, with this assumption, all input data fits in the main memory. However, many modern applications process a large amount of data whose size is greater than the main memory of computers. Therefore, massive data is stored in secondary memory, like disks. In this case, the number of reading/writing from/to the disk is considered as performance criteria, since the time of transferring data between the disk and the main memory is high and usually overcomes the time of processing them in the processor. Modeling the secondary memory and the main memory system is a complex task. The main feature that must be considered is the high access time to the data stored in the secondary memory. Technically, the data transferred between the main memory and disk as blocks. In this paper, we use the standard external memory model that contains one main memory and one disk with the following parameters [2] (see Figure 1):

- *N*: The size of the input data
- *M*: The size of main memory.
- *B*: The size of the disk block.

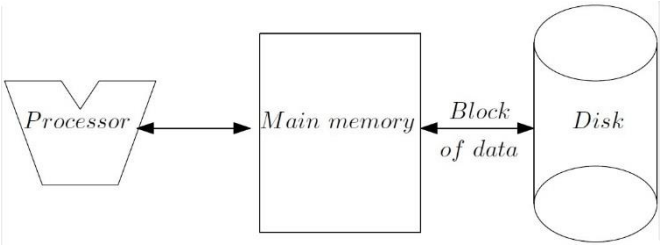

Figure 1: Standard external memory model.

An I/O operation is the operation of reading/writing a block of data from/to the disk. The computation cost in this model is free since the processing is much faster than I/O operation and therefore the amount of time used for processing is negligible, compared to the time spends on I/O operations. So, the complexity of an algorithm in this model is equal to the number of I/O operations that is done during the execution of the algorithm. For example, the complexity of scanning a set of *N* items (stored on the disk) is $O(N/B)$, because N/B blocks of data must be transferred into the main memory.

Our result: In this paper, we consider the problem of sorting a set of *N* real numbers in the external memory model. We propose a progressive algorithm with $O(\log_{M/B} N/B)$ steps which takes $O(N/B)$ I/O operation in each step. The convergence function of our algorithm in step r is $O(\frac{N}{(M/B)^{r/2}})$.

1.2. Related work

One of the fundamental problems that arises in almost all areas of computer science is the sorting problem. Many I/O efficient algorithms have been proposed for the sorting problem and it is difficult to survey all of them. Therefore, we mention some important results about it. Basically, I/O efficient sorting algorithms categorize into two groups. First, the algorithms that use merging approach. In these algorithms, the input data is partitioned into smaller sorted sequences. Then, these sequences are merged until one sequence remains. Second, the algorithms that use distributing approach. In these algorithms, the input data is partitioned into $S_1, \dots, S_k$ such that for any $1 \leq i < j \leq k$ and $x \in S_i, y \in S_j$ we have $x < y$. Then, the sorted order is produced by sorting each of the $S_1, \dots, S_k$ recursively and concatenating the resulting sorted sequences [3]. There is not much difference about I/O complexity of these two approaches. Both algorithms take $O(\frac{N}{B}\log_{M/B}\frac{N}{B})$ I/O to sort the input data [4].

Alewijnse et al. [1] introduced the progressive algorithms in 2014. Also, they proposed two progressive algorithms to find the convex hull of a set of points in the plane and computing the *k*-popular region among a set of given trajectories in the RAM model. Recently, Mesrikhani et.al [5] considered finding the Euclidean minimum spanning tree of *n* points. Their algorithm consists of $O(\log n)$ steps and takes $O(n)$ time in each step. The progressive algorithms are also studied in the experimental context, see [6].

As mentioned before, a progressive algorithm approximates the final solution in each step. It can be shown that there is clear connection between approximation algorithms and progressive algorithms. Suppose we have an approximation algorithm that generates a solution with error $\varepsilon > 0$ in T_{apx} time and the exact solution could be found in $T_{exact}(\varepsilon)$. Then, using the approximation algorithm and for any $k > 1$, there is a *k*-step progressive algorithm with convergence function $O(1/2^r)$ which takes $O(\frac{1}{k} \times (T_{apx}\left(\frac{1}{2^{k-1}}\right) + T_{exact}))$ time in each step [1]. Giesen et al. [7] studied the problem of approximate sorting of *n* numbers in 2009. They considered Spearman's footrule distance as the error function and proposed an algorithm in $O(n \log v(n))$ time with approximation factor *r* where $v(n) = \frac{n^2}{r}$. Also, they showed that, in any comparison model, approximate ranking of *n* numbers with error $O(n^2/v(n))$ needs $n(\min\{\log v(n), \log n\})$ time.

2. Sorting numbers progressively

Let $S = \{a_1, \dots, a_N\}$ be a set of *N* real numbers that we want to sort. To design a progressive algorithm, first, we have to define the partial solutions and the error function. It is clear that the sorted order of S is a permutation of the numbers in *S*. So the partial solution in any step is a permutation of *S* and in step *r* we denote it by S_r. Let S_f be the sorted list of *S* and assume $pos_{S_r}(a_i)$ denotes the position of $a_i \in S_r$ in S_r. The error function f_e is the maximum displacement of any number with respect to S_f (see Figure 2). More precisely:

$$f_e(S_r) = \max_{a_i \in S} \left| pos_{S_f}(a_i) - pos_{S_r}(a_i) \right|.$$

Intuitively, in step r the position of each point is at most $f_e(S_r)$ far from its position in the sorted list.

$$S_f = \{4, 6, 7, 11, 13, 22\}$$
$$S_r = \{6, 4, 22, 13, 7, 11\}$$

Figure 2: The error value of the partial solution S_r is 3.

The idea of our progressive algorithm is based on the I/O efficient distributing algorithm for sorting problem. In each step of the algorithm, $\sqrt{M/B}$ splitting elements are computed. Then the elements of S are distributed into the $\sqrt{M/B}$ unsorted lists $(P_1, \dots, P_{\sqrt{M/B}})$ of roughly equal size such that the elements of P_i are smaller than the elements in P_{i+1}. The splitting elements could be found in $O(\frac{N}{B})$ I/O operation using *Selection algorithm* [4].

Let $P_1 = S$. In generic step r, for each $i = 1, \dots, \left(\frac{M}{B}\right)^{\frac{r-1}{2}}$, we do the following:

- If $|P_i| = M$ then sort P_i.
- Otherwise, sample $\frac{4N}{\sqrt{M/B}}$ elements from P_i as follows:
    1. Create N/M sorted lists that fits into the main memory.
    2. Choose every $\frac{1}{4}\sqrt{M/B}$'th element and assign it to the set H.
- Use *Selection algorithm* M/B times to compute every $\frac{4N}{M/B}$'th element from H and add them to K.
- Distribute P_i into the sets $P_1, \dots, P_{\sqrt{M/B}}$ using the elements of K.
- Report $P_1, \dots, P_{\sqrt{M/B}}$.

To get an upper bound for the partial solution in step r, we use Aggarwal and Vitter [4] idea in analyzing the number of I/O operations for distribution sorting algorithm and get the following lemma.

Lemma 1. If S_r is the partial solution in step r of the algorithm, then $f_e(S_r) \in O(\frac{N}{(M/B)^{r/2}})$.

Proof. To analyze the error of the partial solution S_r, first, we have to determine the cardinality of any P_i. By the algorithm, we have N/M sorted list. The number of element of S (denoted by C) between two arbitrary elements $k_1, k_2 \in K$ is bounded by:

$$C \le C_1 + C_2 + C_3 \quad (1)$$

- C_1: the number of elements of H between k_1 and k_2.
- C_2: the number of elements of S between h_1 and h_2, where $h_1, h_2 \in H$ are two elements between k_1 and k_2.
- C_3: the number of elements of S between k_1 and k_2 but not between h_1 and h_2.

By the algorithm, trivially $C_1 \le \frac{4N}{M/B}$. To get an upper bound for C_2, we do as follows. By the algorithm, the number of elements of S between two consecutive members of H is $\frac{\sqrt{M/B}}{4} - 1$. Since $|H| = \frac{4N}{\sqrt{M/B}}$, we have:

$$C_2 \le \frac{4N}{M/B}\left(\frac{\sqrt{M/B}}{4} - 1\right) = \frac{N}{\sqrt{M/B}} - \frac{4N}{M/B} \quad (2)$$

To analyze the value of C_3, we consider two boundaries that are defined by h_1 and h_2. The number of elements between h_1 and the closest sampled element to h_1 is at most $\frac{\sqrt{M/B}}{4} - 1$. Since we consider two boundaries that are defined by h_1 and h_2, and the total number of boundaries is $\frac{2N}{M}$, we have:

$$C_3 \le \frac{2N}{M}\left(\frac{\sqrt{M/B}}{4} - 1\right) \le \frac{N}{2B\sqrt{M/B}} \le \frac{N}{2\sqrt{M/B}} \quad (3)$$

By Equations (2) and (3), we can rewrite Equation (3) as follows:

$$C \le \frac{4N}{M/B} + \frac{N}{\sqrt{M/B}} - \frac{4N}{M/B} + \frac{N}{2\sqrt{M/B}}$$
$$\le \frac{N}{\sqrt{M/B}} + \frac{N}{2\sqrt{M/B}} = \frac{3}{2} \times \frac{N}{\sqrt{M/B}}$$

So, the cardinality of any P_i is $O(\frac{N}{\sqrt{M/B}})$. In the next steps of the algorithm, each P_i is partitioned into $\sqrt{M/B}$ subsets. Consequently, the size of P_i in step number r is $O(\frac{N}{(M/B)^{r/2}})$ and the maximum displacement in any of P_i is no more than $O(\frac{N}{(M/B)^{r/2}})$ and the proof is completed. ■

Theorem 1. There exists a progressive algorithm for sorting a set of N real numbers with $O(\log_{M/B} N/B)$ steps that takes $O(N/B)$ I/O operations in each step. The convergence function of this algorithm in step r is $O(\frac{N}{(M/B)^{r/2}})$ with respect to the error function $f_e(S_r)$.

Proof. The convergence function can be obtained directly from Lemma 1. To analyze the number of I/O operations in each step of the algorithm, we do as follows. Creating the set H could be done by scanning all element of each P_i. So, this takes $\sum_{i=1}^{\left(\frac{M}{B}\right)^{r/2}} \frac{|P_i|}{B}$ I/Os in step r which is at most $O(N/B)$, since $\sum_{i=1}^{\left(\frac{M}{B}\right)^{r/2}} |P_i| = N$. Calling *Selection algorithm* $\sqrt{M/B}$ times takes $O(N/B)$ I/Os, since the cardinality of H is $\frac{4N}{\sqrt{M/B}}$. Finally distributing the elements in the corresponding set takes $O(N/B)$ I/Os by simply scanning all elements. Consequently, each step of the algorithm takes $O(N/B)$ I/O operations. The algorithm continues until $|P_i| = M$ .By Lemma 1, we know that $|P_i| = \frac{N}{(M/B)^{r/2}}$ in step r. So, the

number of steps of the algorithm is $O(\log_{M/B} N/B)$ and the proof is completed. ■

3. Conclusion

In this paper, we consider the problem of sorting a massive set of real numbers in the external memory model. We design a progressive algorithm for sorting N real numbers with $O(\log_{M/B} N/B)$ steps, which takes $O(N/B)$ I/O operations in each step. Our algorithm generates a partial solution in step r of the algorithm with the error value $O(\frac{N}{(M/B)^{r/2}})$. Designing a progressive algorithms for other problems with massive input data is an interesting future work.

References

[1] S. P. A. Alewijnse, T. M. Bagautdinov, M. de Berg, Q. W. Bouts, A. P. ten Brink, K. Buchin, M. A. Westenberg, "Progressive geometric algorithms," Proc, *Thirtieth Annual Symposium on Computational Geometry, SOCG'14, ACM, New York, NY, USA*, pp. 50-59, 2014.

[2] MA. Aggarwal, B. Alpern, A. Chandra, M. Snir, "A model for hierarchical memory," Proc, *Nineteenth annual ACM symposium on Theory of computing, ACM*, pp. 305-314, 1987.

[3] C. A. Hoare, "Quicksort," *The Computer Journal*, vol. 5, no. 1, pp. 10-16, 1962.

[4] A. Aggarwal, J. Vitter, "The input/output complexity of sorting and related problems," *Communications of the ACM* vol. 31, no. 9, pp. 1116-1127, 1988.

[5] A. Mesrikhani, M. Farshi, M. Davoodi, "Progressive algorithm for Euclidean minimum spanning tree," Proc, *1st Iranian Conference on Computational Geometry*, pp. 29-32, 2018.

[6] R.H. Li, L. Qin, J. X. Yu, R. Mao, "Efficient and progressive group steiner tree search," Proc, *International Conference on Management of Data, ACM*, pp. 91-106, 2016.

[7] J. Giesen, E. Schuberth, M. Stojakovic, Approximate sorting, Fundamenta Informaticae vol. 90, no. 1-2, pp. 67-72, 2009.

Amir Mesrikhani received his B.S. degree in computer science from Kharazmi University, Tehran, Iran, in 2011, and his M.S degrees in Institute for Advanced Studies in Basic Sciences (IASBS), Zanjan, Iran in 2014. Now he is a Ph.D. student in the Department of Computer Science, Yazd University, Yazd, Iran. His research interests are computational geometry, massive data and approximation algorithm.

Email: mesrikhani@stu.yazd.ac.ir

Mohammad Farshi received his BSc in Computer Science from Yazd University, Yazd, Iran in 1996 and his MSc in Pure Mathematics from Shiraz University, Shiraz, Iran, in 1999, and his PhD in Computer Science from Eindhoven University of Technology, Eindhoven, The Netherlands, in 2008. Between 1999 and 2004, he worked at Yazd University as an instructor, and since 2009 he has been an Assistant Professor at the same university. His research interests are computational geometry, geometric spanners and algorithms.

Email: mfarshi@yazd.ac.ir

Paper Handling Data:

Submitted: 10/6/2018
Received in revised form: 1/7/2018
Accepted: 9/7/2018
Corresponding author: Mohammad Farshi
Combinatorial and Geometric Algorithms Lab,
Department of Mathematical Sciences, Yazd University,
Yazd, Iran.