PAPER Special Issue on Discrete Mathematics and Its Applications

A Geometric Fitting Problem of Two Corresponding Sets of Points on a Line

Hiroshi IMAI†, *Member*

SUMMARY This paper gives an $O(n \log n)$-time algorithm for the following problem:

$$\min \sum_{i=1}^{n} |x_i - a_i| \qquad \text{s. t. } x_1 \leqq x_2 \leqq \cdots \leqq x_n$$

where $a_i (i=1, \cdots, n)$ are given constants and $x_i (i=1, \cdots, n)$ are variables. There has been given an $O(n^2)$-time incremental algorithm[2], and we improve it by using the heap as a data structure and modify the incremental algorithm partly. This problem is a kind of the geometric fitting problem between two corresponding sets of points on a line which is related to some VLSI layout design problem.

1. Introduction

This paper considers the following problem:

$$\min \sum_{i=1}^{n} |x_i - a_i| \qquad \text{s. t. } x_1 \leqq x_2 \leqq \cdots \leqq x_n$$

where $a_i (i=1, \cdots, n)$ are given constants and $x_i (i=1, \cdots, n)$ are variables. For a set of n fixed points p_i on the x-axis such that the x-coordinate of p_i is a_i, this problem locates n points $q_1, q_2, \cdots, q_n$ (the x-coordinate of q_i is x_i) on the x-axis from left to right (i. e., $x_1 \leqq x_2 \leqq \cdots \leqq x_n$) so that the sum of the distances between p_i and q_i is minimized. This problem is thus a kind of the geometric fitting problem between two corresponding sets of points on a line which is related to some VLSI layout design problem[2],[3].

An $O(n^2)$-time incremental algorithm is given in Ref. (2). This paper gives an $O(n \log n)$-time algorithm for this problem by using the heap as a data structure and modify the incremental algorithm partly.

The problem treated here is a very special case of the linear programming problem. When applied to VLSI layout design, this problem might be too restrictive. However, formulating the problem of removing jogs in the VLSI compaction problem[3] in a similar way to this problem would be useful, since then the interior point algorithm for linear programming with planar structures[1] can be applied.

Manuscript received November 19, 1990.

† The author is with the Faculty of Science, The University of Tokyo, Tokyo, 113 Japan.

2. Preliminaries

We here consider two special cases of the problem. When $a_1 \leqq a_2 \leqq \cdots \leqq a_n$, this problem is trivially solved, since then $x_i = a_i$ is a unique optimum solution. However, if $a_i (i=1, \cdots, n)$ are not in nondecreasing order, the problem is never trivial.

Next consider the problem where all $x_i (i=1, \cdots, n)$ are set to be equal. The Weber problem for a set of points is to find a center point that minimizes the sum of the distances of the center point with points in the set. Although the two-dimensional Weber problem is very hard to solve rigorously, the one-dimensional problem is easy to solve. For the one-dimensional Weber problem, the following is well known.

[Lemma 2. 1] An optimum solution of the problem of minimizing $\sum_{i=1}^{n} |x - a_i|$ is the median among $a_i (i=1, \cdots, n)$. □

For simplicity, suppose that $a_i (i=1, \cdots, n)$ are distinct to one another. If n is odd, the median is the $\lceil n/2 \rceil$th largest value among a_i. If n is even, $(n/2)$th and $(1+n/2)$th values (and any value between them) may be considered to be the median.

To avoid the degeneracy for even n, we regard that

$$|x - a_i| = \begin{cases} (1+\varepsilon_i)(x - a_i) & x - a_i \geqq 0 \\ -(x - a_i) & x - a_i < 0 \end{cases}$$

for sufficiently small positive number ε_i. Then, the optimum solution is uniquely determined, and is the $\lceil n/2 \rceil$th value among a_i for any n. We will simply call this value the median.

3. Incremental Algorithm

In this section, we describe the incremental algorithm given in Ref. (2). In that paper, some of inequalities among x_i are set to hold with equalities. In this case, the problem is stated as follows:

$$\min \sum_{j=1}^{m} \sum_{i \in I_j} |x_j - a_i| \qquad \text{s. t. } x_1 \leqq x_2 \leqq \cdots \leqq x_m \qquad \text{(P)}$$

where $a_i (i=1, \cdots, n)$ are given constants, $X_j (j=1, \cdots, m)$ are variables, and $\{I_j \mid j=1, \cdots, m\}$ is a partition of

$\{1, 2, \cdots, n\}(m \leqq n)(I_j \neq I_{j'}$ for $j \neq j'$ and $\bigcup_{j=1}^{m} I_j = \{1, 2, \cdots, n\})$. We will consider the problem in this general form.

For a subset I of $\{1, 2, \cdots, n\}$, define med(I) to be the median among $a_i (i \in I)$. During the incremental algorithm, adjacent sets $I_k, I_{k+1}, \cdots, I_{k'}$, among $I_j (j=1, \cdots, m)$ are merged into a set I, and all x_j associated with I (i.e., $x_k, x_{k+1}, \cdots, x_{k'}$) are conceptually set to med(I). To maintain the adjacency relations among sets I_j, we use a list L of sets $I_j \cdot L$ is initially empty. During the algorithm, for each set $\tilde{I}$ in the list L, we maintain the value of med$(\tilde{I})$.

The incremental algorithm consists of m stages. In the first stage, it starts with computing the value b_1 of med(I_1). $x_1 = b_1$ is an optimum solution for the problem consisting of only the set I_1. Add I_1 to the empty list L.

In the jth stage $(j \geqq 2)$, the algorithm tries to add the constraints concerning I_j to the current optimum solution for $I_1, \cdots, I_{j-1}$ which has been computed already to obtain an optimum solution for $I_1, \cdots, I_{j-1}, I_j$. First, add I_j at the tail of the list L, and set $I = I_j$.

The jth stage then iterates the following. Compute the value of med(I). Let I' be the predecessor of I in the list L (if $I = I_j$ and I_{j-1} has not yet been merged, $I' = I_{j-1}$). For this I', the value of med(I') has been computed and recorded. If med$(I') \leqq$ med(I), we have computed an optimum solution for $I_1, \cdots, I_j$ (the optimum is obtained by, for each set $\tilde{I}$ in the list L, setting x_j associated with $\tilde{I}$ to med$(\tilde{I})$), and proceed to the $(j+1)$st stage. Otherwise, merge I and I' into one (accordingly update the list L), and, regarding the merged set as I, repeat this procedure for this updated I. Here, if the predecessor of the merged set in L is empty, proceed to the $(j+1)$st stage.

The algorithm halts after the mth stage. An optimum solution is obtained by setting x_j associated with $\tilde{I}$ in the list L at the end to med$(\tilde{I})$, the value of which is maintained for each $\tilde{I}$.

In this algorithm, for any set $I = I_k \cup I_{k+1} \cup \cdots \cup I_{k'}$ in the list L, $x_j (j = k, k+1, \cdots, k')$ are constrained to be identical. Merging I and I' at some stage in the above algorithm corresponds to adding a new constraint that x_j associated with I and $x_{j'}$ associated with I' should also be identical. We can show that adding this constraint to the problem does not increase the optimum objective function value, and the validity of this incremental algorithm follows (see Ref. (2)).

[Lemma 3.1] The incremental algorithm correctly finds an optimum solution. □

[Example 3. 1] We here give an example showing how the incremental algorithm works. Consider the problem with

$n=9,\ m=4,$

$I_1=\{1, 2, 3\},\ I_2=\{4\},\ I_3=\{5, 6\},\ I_4=\{7, 8, 9\}$

$a_7 < a_8 < a_9 < a_1 < a_5 < a_2 < a_6 < a_3 < a_4$

It should be noted that a total order among a_i suffices to determine the optimum solution, which is seen from the incremental algorithm itself.

In this example, x_1 is set to a_2 after the first stage. Then, x_2 is set to a_4. Since $a_5 = \text{med}(I_3) < \text{med}(I_2) = a_4$, I_2 and I_3 are merged, and x_2 and x_3 are set to $a_6 = \text{med}(I_2 \cup I_3)$. In the fourth stage, $a_8 = \text{med}(I_4) < \text{med}(I_2 \cup I_3)$, so that I_4 is merged with $I_2 \cup I_3$. Then, $a_9 = \text{med}(I_2 \cup I_3 \cup I_4) < \text{med}(I_1)$, and $I_2 \cup I_3 \cup I_4$ is merged with I_1. Thus all the sets are merged in this case, and the optimum solution is $x_j = a_5 (j=1, \cdots, 4)$. □

The median of a set can be found in time linear to the size of the set. Using this algorithm, it is easy to show that this incremental algorithm can be implemented so as to run in $O(mn)$ time[2].

4. Improved Algorithm

We now consider how to implement the incremental algorithm so as to run in $O(n \log n)$ time. First observe the following.

[Lemma 4. 1] Suppose that a set I is in the list L just after some stage. Then, the values, among $a_i (i \in I)$, which are greater than med(I) will not become med(I') for any set I' in the list L.

(Proof) After the stage, I will be scanned again when, for its successor $\tilde{I}$ in L, med$(\tilde{I}) <$ med(I). Then, I and merged into one. Since med$(\tilde{I}) <$ med(I), the median for the merged set is not greater than med (I), and the values $a_i (i \in I)$ greater than med(I) cannot become the median med$(I \cup \tilde{I})$ for the merged set $I \cup \tilde{I}$.

After I is merged with $\tilde{I}$, the merged set may be further merged with its predecessor I' when med$(I \cup \tilde{I}) <$ med(I') (I' was the predecessor of I in L just after the stage, and is now that of $I \cup \tilde{I}$ in L after merging I and $\tilde{I}$). In such a case, the median for the set after merging I' with $I \cup \tilde{I}$ does not exceed the median for the original I'. Since I' was the predecessor of I in L just after the stage, med$(I') \leqq$ med(I) for these I and I'. Hence, the values among a_i for i in the original I which are greater than med(I) cannot become the median for the set in L.

Using this argument inductively proves the lemma. □

[Remark 4. 1] During the jth, stage, the values a_i $(i \in I_j)$ greater than the median med(I_j) for the original I_j may become the median for a set in L. For instance, in the third stage in Example 3. 1, $a_6 (> \text{med}(I_3) = a_5)$ becomes the median for the merged set $I_2 \cup I_3$. Similar for merged sets obtained in the process of the jth stage. During the fourth stage in the same example, $I = I_2 \cup I_3 \cup I_4$ is a temporal merged set, and $a_9 = \text{med}(I) < a_5$, while, at the end of the fourth stage, $a_5 = \text{med}(I_1 \cup I)$. □

We now present a modified incremental algorithm with early merging. To maintain and compute the value $\mathrm{med}(I)$ for sets I in L efficiently, we use mergeable heaps. For a set I in L, we maintain a heap consisting of some of $a_i(i\in I)$. Just after any stage of the algorithm, the elements of the heap for I in L are less than or equal to $\mathrm{med}(I)$, and the median is the maximum in the heap. Also, the number of $a_i(i\in I)$ which are not contained in the heap is maintained at any time. Just after the stage, this number is the number of $a_i(i\in I)$ greater than $\mathrm{med}(I)$.

In the jth stage, $I=I_j$ is handled. We add I to the list L, and, for this $I=I_j$, construct a heap consisting of all $a_i(i\in I_j)$.

Then the jth stage iterates the following. Let I' be the predecessor of I in L. For this I', the value of $\mathrm{med}(I')$ is already computed. While the maximum of the heap for I is greater than $\mathrm{med}(I)$ and $\mathrm{med}(I')$, we iterate to delete the maximum from the heap. Since the number of $a_i(i\in I)$ which are not contained in the heap is maintained, we know how many elements should be deleted from the heap to find $\mathrm{med}(I)$.

In so doing, we compare the maximum value of the heap for I with $\mathrm{med}(I')$. If all the values compared with $\mathrm{med}(I')$ are geater than or equal to $\mathrm{med}(I')$, the maximum of the current heap becomes $\mathrm{med}(I)$ and $\mathrm{med}(I)$ is not less than $\mathrm{med}(I')$. In this case, we proceed to the $(j+1)$st stage. Otherwise, some extracted maximum value, which is greater than or equal to $\mathrm{med}(I)$ for the current set I, is found to be less than $\mathrm{med}(I')$. Then, it is seen at this point that $\mathrm{med}(I)<\mathrm{med}(I')$, and hence I' will be further merged to the current set I. At this point, we immediately merge the heap for I' with the current heap for I even before the median for the current set is found (this is *early merging*), and repeat this procedure for this new merged set $I:=I\cup I'$. When the predecessor of the new set I in the list L becomes empty, we just find the median for the current set, and proceed to the next stage.

We now consider the validity of this modified algorithm. As shown in Lemma 4. 1, for the original I' above, the values $a_i(i\in I')$ which are greater than $\mathrm{med}(I')$ cannot become the median for any merged set. However, the values $a_i(i\in I_j)$ greater than $\mathrm{med}(I_j)$ for the original $I=I_j$ may become the median in some merged set ; a similar thing may occur for merged sets obtained during the jth stage (cf. Remark 4. 1). In the above algorithm, all $a_i(i\in I_j)$ are first collected into a heap, and therefore are checked for the median correctly. Also, for any merged set appearing during the jth stage, we do not delete, from the heap for the set, any value which may become the median for some merged set in future by virtue of early merging. Hence, the medians for sets in L can be maintained using heaps. We thus obtain the following lemma.

[Lemma 4. 2] At the end of any stage, the list L obtained by the original incremental algorithm and that obtained by the above algorithm with early merging are the same. □

Then, from Lemma 3. 1, the validity of this modified algorithm follows.

We now analyze the time complexity of this modified algorithm with the data structure. Each element is removed from the heap at most once, and the number of times two heaps are merged into one is at most $m-1\leqq n-1$. Finding and deleting the maximum in the heap and merging two heaps can be done in $O(\log n)$ time (e. g., see Ref. (4)). Hence, the total complexity is bounded by $O(n\log n)$.

[Theorem 4. 1] The geometric fitting problem (P) of two corresponding sets of n points on a line can be solved in $O(n\log n)$ time. □

5. Concluding Remarks

In this paper we have shown that the simple incremental algorithm for some special one-dimensional geometric fitting problem can be implemented so as to run in $O(n\log n)$ time. It is left open whether this problem can be solved in linear time, say by the prune-and-search method.

As mentioned in the introduction, the problem treated here is a very special case of the linear programming problem, and might be too restrictive to apply it to a general VLSI layout design problem. However, formulating the problem of removing jogs in the VLSI compaction problem, considered in Ref. (3), in a similar way to this problem would be very useful, since then the interior point algorithm for linear programming with planar structures[1] can be applied. This issue will be discussed elsewhere.

Acknowledgement

This research was supported in part by the Grant-in-Aid of the Ministry of Education, Science and Culture of Japan and by the Inamori Foundation.

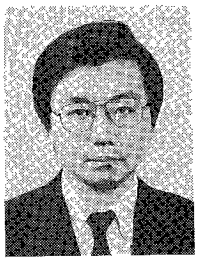
References

(1) Imai H. and Iwano K. : "Efficient Sequential and Parallel Algorithms for Planar Network Flow", Proceedings of the SIGAL International Symposium on Algorithms, Lecture Notes in Computer Science, **450**, pp. 21-30, Springer-Verlag, Heidelberg (1990).

(2) Ohmura M., Wakabayashi S., Miyao J. and Yoshida N. : "Improvement of One Dimensional Module Placement in VLSI Layout Design", Trans. IEICE, **J73-A**, 11, pp. 1858-1866 (1990).

(3) Sato M., Yamamoto W., Nakajima N. and Ohtsuki T. : "A Chip Compaction Algorithm with Jog Insertion", Technical Report SIGAL 90-16-11, Information Processing Society of Japan (1990).

(4) Tarjan R. E. : "Data Structures and Network Algorithms", SIAM, Philadelphia (1983).



Hiroshi Imai was born in November 21, 1958. He obtained B. Eng. in Mathematical Engineering, and M. Eng. and D. Eng. in Information Engineering, University of Tokyo in 1981, 1983 and 1986, respectively. In 1986–1990, He was an associate professor of Department of Computer Science and Communication Engineering, Kyushu University. Since 1990, he has been an associate professor at Department of Information Science, University of Tokyo. His research interests include algorithms, computational geometry, and optimization. He is a member of IPSJ and ACM.