アルゴリズム 18－5
（1990. 11. 22）

対応する点集合間の1次元幾何的マッチング問題

今井浩
東京大学理学部情報科学科

VLSI のレイアウト設計などにおいて、2つの対応する点集合の間のマッチングを求める問題が現われる。VLSI レイアウトにおけるモジュール配置で、平行な2直線上に配置されているモジュール集合の間の配線長を最小にすることを簡単に定式化すると、次の1次元の幾何的なマッチング問題になる。

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

ここで、a_i $(i = 1, \ldots, n)$ は与えられた定数で、x_i $(i = 1, \ldots, n)$ は変数である。この問題は特殊な線形計画問題である。この問題に対しては、既に $O(n^2)$ の手間のアルゴリズムが与えられている[2]。本稿では、データ構造としてヒープを用い、またアルゴリズムの一部を変えることにより、この問題が $O(n \log n)$ の手間で解けることを示す。

One-Dimensional Geometric Fitting Problem of Two Corresponding Sets of Points

Hiroshi Imai
Department of Information Science, Faculty of Science
University of Tokyo, Bunkyo-ku, Tokyo 113, Japan

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

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

where a_i $(i = 1, \ldots, n)$ are given constants and x_i $(i = 1, \ldots, n)$ are variables. There has been given an $O(n^2)$-time incremental algorithm, and we improve it by using heaps 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 geometric problem:

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

where a_i $(i = 1, \ldots, n)$ are given constants and x_i $(i = 1, \ldots, n)$ are variables. 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 [2,3].

An $O(n^2)$-time incremental algorithm is given in [2]. This paper gives an $O(n \log n)$-time algorithm for this problem by using heaps 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, scince then the interior point algorithm for linear programming with planar structures [1] can be applied.

2. Preliminaries

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

Next consider the problem where all x_i $(i = 1, \ldots, n)$ 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 $(i = 1, \ldots, n)$. $\square$

For simplicity, suppose that a_i $(i = 1, \ldots, n)$ are distinct to one another. If n is odd, the median is the $\lfloor n/2 \rfloor$th largest value among a_i. If n is even, $(n/2)$th and $(1 + n/2)$th values are both considered to be the median in most cases, and any value between them is optimum for the problem.

To avoid the degeneracy for even n, we suppose that

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

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. We will simply call this value the median.

3. Incremental Algorithm

In this section, we describe the incremental algorithm given in [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:

$$\begin{aligned} &\min \sum_{j=1}^{m} \sum_{i \in I_j} |x_j - a_i| \\ &\text{s.t. } x_1 \le x_2 \le \cdots \le x_m \end{aligned} \tag{P}$$

where a_i $(i = 1, \dots, n)$ are given constants, x_i $(i = 1, \dots, m)$ are variables, and $\{I_j \mid j = 1, \dots, m\}$ is a partition of $\{1, 2, \dots, n\}$ $(m \le n)$ $(I_j \ne I_{j'}$ for $j \ne j'$ and $\bigcup_{j=1}^{m} I_j = \{1, 2, \dots, n\})$. We will consider the problem in this general form.

During the incremental algorithm, adjacent sets among I_j are merged into one and x_i for i in the merged set are set to be equal to one another. To maintain the adjacency relations among I_j, we use a list L which is initially empty.

The incremental algorithm consists of m stages. In the first stage, it starts with computing the $\lceil |I_1|/2 \rceil$th value b_1 among a_i $(i \in I_1)$ and then setting $x_1 = b_1$. This is a optimum solution for the problem consisting of only the set I_1. Add I_1 to the empty list L.

In the jth stage $(j \ge 2)$, the algorithm adds the constraints concerning I_j to the current optimum solution for $I_1, \dots, I_{j-1}$ which has been computed already. Add I_j at the tail of the list L. Let b_j be the $\lceil |I_j|/2 \rceil$th value among a_i $(i \in I_j)$. Let I' be the predecessor of I_j in the list L (if I_{j-1} has not yet been merged, $I' = I_{j-1}$). Let b' be the value for x_i $(i \in I')$ in the current solution (b' is the median among a_i $(i \in I')$).

If $b' \le b_j$, set $x_i = b_j$ for $i \in I_j$ (this is the optimum solution for $I_1, \dots, I_j$), and proceed to the $(j+1)$st stage. Otherwise, merge I_j and I' into one (accordingly update the list L), and regarding the merged set as I_j repeat this procedure for this updated I_j. 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.

For the validity of this incremental algorithm, see [2]. We here give an example. Consider the problem with

$$\begin{aligned} &n = 9, \qquad m = 5, \\ &I_1 = \{1,2,3\},\ I_2 = \{4\},\ I_3 = \{5\},\ I_4 = \{6,7\},\ I_5 = \{8,9\} \\ &a_6 < a_8 < a_9 < a_1 < a_2 < a_7 < a_3 < a_4 < a_5 \end{aligned}$$

Surprisingly, a total order among a_i suffices to determine the optimum solution.

In this example, $x_1 = x_2 = x_3$ is set to a_2 after the first stage. Then x_4 and x_5 are set to a_4 and a_5 in the second and third stages, respectively. For I_4, the median among a_6 and a_7 is a_6, and this value is compared with the current value a_5 of x_5 of the predecessor I_3 of I_4 in L. Since $a_6 < a_5$, I_3 and I_4 are merged. For this merged set, the median is a_7. This value is further compared with the value a_4 for the predecessor set I_2 of the merged set in L. Again, $a_7 < a_4$, and so I_2 is merged into $I_3 \cup I_4$. For this merged set, the median is a_7, and is greater than the value a_2 for the predecessor set I_1 in L. The fourth stage thus finishes. Now, I_5 is processed, and its median a_8 is compared with the median a_7 in the predecessor in L. Since $a_8 < a_7$, these two sets are merged. The median becomes a_9, which is less than a_2, and these are further merged with I_1. Thus, finally all the sets are merged in this example, and the optimum solution is $x_i = a_2$ $(i = 1, \ldots, 5)$.

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 to run in $O(n \log n)$ time. First observe the following.

Lemma 4.1. Suppose a set I is in the list L just after some stage. Then, the values in I greater than the median of I will not become the median of any set in the list L.

Proof: After the stage, I will be scanned again when its successor $\tilde{I}$ in L has the median $\tilde{b}$ smaller than the median b of I. Then, I and $\tilde{I}$ are merged into one. Since $\tilde{b} \le b$, the median of the merged set is not greater than b, and so the values in I greater than b cannot become the median of the merged set in L.

After I is merged with $\tilde{I}$, the merged set may be further merged with its predecessor I' when the median of the merged set is less than or equal to the median of I'. In such a case, the median of the set after merging I' does not exceed the median of the original I'. Since the original I' is the ancestor of the original I in L, its median is less than the median of the original I. Hence, the values in the original I greater than b cannot become the median of the set in L.

Using this argument inductively proves the lemma. □

Note that, during the jth stage, the values greater than the median of I_j may become the median of a set in L (e.g., a_7 in the above example).

We now present a modified incremental algorithm with *early merging*. To maintain and compute the median of sets in L efficiently, we use mergeable heaps. For a set I in L, we maintain a heap consisting of values less than or equal to the median in I. The median is the maximum in the heap. Also, the number of elements in I not contained in the heap is kept (the number of elements greater than the median).

In the jth stage, $I = I_j$ is handled. We first compute the median b of I by a linear-time algorithm. We also add I to the list L, and, for this $I = I_j$, construct a heap consisting of all elements of I_j. Let I' be the predecessor of I in L and b' the median of I'. Suppose $b < b'$ (otherwise, we are done). Then I and I' have to be merged. We first merge two heaps for I and I' into a heap h. Let I'' be the predecessor of I' in L and b'' be the median of I''. We repeat to extract the maximum from the heap h to find the median of the merged set. Since we know the number of elements greater than the median in I', we know how many elements should be extracted from the heap (simultaneously we maintain the number of elements greater than the median in the merged set).

In so doing, we compare each extracted maximum with b''. If all the extracted maximums are greater than or equal to b'' and the median of the merged set is already found, the median of the merged set is not less than b'' and we proceed to the $(j+1)$st stage.

Otherwise, some extracted maximum, which is greater than or equal to the median of the merged set, is found to be less than b''. Then, it is seen at this point that the median of the merged set is less than b'', and hence I'' will be further merged to the current set. At this point, we merge the heap for I'' with the heap h even before the median of the current set is found (this is *early merging*), and repeat this procedure for the merged set, including I'', and the predecessor I''' of I''.

As shown in Lemma 4.1, the elements greater than the median in the set I', I'', I''' above are greater than the median of the merged sets and cannot become the median. The elements greater than the median in $I = I_j$ may become the median in the merged set. In the above algorithm, all the elements of I_j is first collected into a heap and so all the elements of I are checked for the median correctly. The validity of this modified algorithm follows.

The above algorithm improves the simple incremental algorithm in a point that, while computing the median of the merged set of I and I', whether I'' should be merged further is checked simultaneously and that I'' may be merged correctly before the median of the merged set of I and I' is computed. This enables us to eliminate extracted maximums from the merged set of I and I' from further consideration.

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 $n - 1$. Finding and deleting the maximum in the heap and merging two heaps can be done in $O(\log n)$ time (e.g., see [4]). Hence, the total complexity is bounded by $O(n \log n)$.

Theorem 4.1. The problem (P) can be solved in $O(n \log n)$ time. $\square$

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 pune-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 desing problem. However, formulating the problem of removing jogs in the VLSI compaction problem, considered in [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.

Acknoledgment

This research was supported in part by the Grant-in-Aid of the Ministry of Education, Science and Culture of Japan and the Okawa Institute of Information and Telecommunication.

References

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

[2] M. Ohmura, K. Yokoyama, S. Wakabayashi, J. Miyao and N. Yoshida: On Improvement of VLSI Module Placement Based on Critical Nets (in Japanese). *COMP88-52*, IEICE, 1988.

[3] M. Sato, W. Yamamoto, N. Nakajima and T. Ohtsuki: A Chip Compaction Algorithm with Jog Insertion (in Japanese). *SIGAL 90-16-11*, IPSJ, 1990.

[4] R. E. Tarjan: *Data Structures and Network Algorithms.* SIAM, Philadelphia, 1983.