On a geometric approach to the segment sum problem and its generalization

Asish Mukhopadhyay *† Eugene Greene *‡

Abstract

Given a sequence of n real numbers $a_1, a_2, a_3, \ldots, a_n$, the *maximum segment sum* problem is that of determining indices i and j $(1 \leq i \leq j \leq n)$ such that the sum $s(i,j) = a_i + a_{i+1} + \ldots + a_j$ is a maximum. Monotone matrices were shown to be remarkably effective in solving several geometric optimization problems. The surprise is that it can also be applied to the above problem as we show here. Recently, there was a breakthrough in obtaining an $O(n \log n)$ algorithm for the kth smallest segment sum problem by exploiting a connection of this problem to the well-known slope selection problem. In this paper we show that this problem can also be solved within the same time bounds in the simpler framework of expander graphs.

1 Introduction

Given a sequence of n real numbers $a_1, a_2, a_3, \ldots, a_n$, the *maximum segment sum* problem is that of determining indices i and j $(1 \leq i \leq j \leq n)$ such that the sum $s(i,j) = a_i + a_{i+1} + \ldots + a_j$ is a maximum. This problem was introduced by Jon Bentley in his CACM column (#8) on Programming Pearls. He described an elegant linear time algorithm due to J. B. Kadane [5].

Monotone matrices have been used to solve a variety of geometric optimization problems [1]. In this paper, we show how it can applied to solve the largest segment sum problem in linear time. In fact, this scheme allows us to find the largest segment sum for all start positions in the array, as well as the largest segment sum for a segment length that lies between specified length-parameters l and u, within the same time bounds.

Lately, there has been a flurry of attempts to solve a more general version of this problem, namely, finding the kth smallest segment sum, [6], [3], [4], culminating in an $O(n \log n)$ algorithm by Lin and Lee[13], who also gave an $O(n \log n + k)$ algorithm for enumerating the k smallest segment sums. This breakthrough was obtained by reducing this problem to a geometric problem. Interestingly enough, the problem of finding a segment of maximum density whose length lies between input length parameters l and u was solved by Kim [12] also by reduction to a geometric problem.

While the algorithm of Lin and Lee [13] is of theoretical interest, it opens up the prospect that a simpler algorithm may be possible, and this is the principal motivation behind this paper. We show that an expander-graph based framework can also be used for finding the kth smallest segment sum. In our opinion, this makes things simpler, though not breathtakingly so as is our goal.

This paper is structured as follows. In the next section we briefly describe Kadane's scheme, and the monotone-matrix based approach in the next. In Section 4, we briefly outline Lin and Lee's algorithm, and in the following the expander graph based modification of this algorithm. In the sixth and last section we conclude.

2 Kadane's scheme

Kadane's scheme is based on the clever observation that a maximum segment sum cannot have a prefix with a negative sum, or, for that matter a suffix with a negative sum.

It finds the *start*, *end*, and *maxSum* of a maximum segment sum, using three variables - two index variables i and j, with $i \leq j$ always and a *currentSum* which is the segment sum from i to j. As j sweeps over the array, the values of *start*, *end* and *maxSum* are updated whenever *currentSum* exceeds *maxSum*. Whenever *currentSum* becomes negative, the variable i jumps to $j + 1$, since all segment sums with a start value in $[i..j]$ would have a negative prefix.

The time complexity of this scheme is clearly in $O(n)$.

3 Monotone matrix approach

An $n \times n$ matrix of reals is *monotone* if the maximum entry in row i occurs in the same column or in a column to the right of the column in which the maximum entry

*School of Computer Science, University of Windsor, Windsor, Canada

†asishm@cs.uwindsor.ca, Partially supported by an NSERC operating grant

‡greene6@uwindsor.ca, Supported by an NSERC USRA

in row $i-1$ occurs. See Example 1.

A matrix is said to be *totally monotone* if for any 2×2 submatrix $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$, it is not simultaneously possible that $a < b$ and $c > d$.

Let M be an $n \times n$ matrix such that $M[i][j] = s(i,j)$ when $i \leq j$, and $M[i][j] = -\infty$ when $i > j$.

Fact 1 *M is a totally monotone matrix.*

Proof. Let indices i, j, k, and l be such that $1 \leq i < k \leq j < l \leq n$. Assume $M[i][j] < M[i][l]$. If $M[i][j] = -\infty$ then $M[k][j] = -\infty$ as well, and so $M[k][j] \leq M[k][l]$. Otherwise, $a_i + \ldots + a_k + \ldots + a_j < a_i + \ldots + a_k + \ldots + a_j + \ldots + a_l$. So $a_k + \ldots + a_j < a_k + \ldots + a_j + \ldots + a_l$, and hence $M[k][j] \leq M[k][l]$. □

By precomputing all prefix sums $P[i] = a_1 + a_2 + \ldots + a_i$ (we define $P[0] = 0$), we can compute an $M[i][j]$ in constant time as $M[i][j] = P[j] - P[i-1]$ for $i \leq j$.

Now we can use the monotone matrix searching results of [1] to determine the largest segment sum in each row in $O(n)$ time. Thus we have a largest segment sum beginning at a given index i ($1 \leq i \leq n$), while the largest of these is a maximum segment sum.

Example 1 *Consider the sequence 5, -10, 6, -10, 7, -10, 8. The monotone matrix, M, corresponding to this sequence is:*

	1	*2*	*3*	*4*	*5*	*6*	*7*
1	*5*	*-5*	*1*	*-9*	*-2*	*-12*	*-4*
2	-∞	*-10*	*-4*	*-14*	*-7*	*-17*	*-9*
3	-∞	-∞	*6*	*-4*	*3*	*-7*	*1*
4	-∞	-∞	-∞	*-10*	*-3*	*-13*	*-5*
5	-∞	-∞	-∞	-∞	*7*	*-3*	*5*
6	-∞	-∞	-∞	-∞	-∞	*-10*	*2*
7	-∞	-∞	-∞	-∞	-∞	-∞	*8*

This approach allows us to solve a slightly more general version of the segment sum problem. We can find the maximum segment sum when the length of such a segment is restricted to lie between l and u, where $1 \leq l \leq u \leq n$, by only considering entries along the diagonals defined by $j - i = l, l+1, \ldots, u$ and setting all other entries to $-\infty$.

4 Lin-Lee's algorithm for kth smallest segment sum

The Lin-Lee algorithm, for $k \geq n$, uses a battery of very powerful tools - Megiddo's parametric search [16], Cole's technique of slowing down a sorting network [7], as well as an approximate counting technique by Cole again [8] adapted to this problem.

The original contribution of the Lin-Lee paper is the reduction of this purely combinatorial optimization problem to a slight variation of the well-known and well-researched slope selection problem. The latter problem is to select a line whose slope is the kth smallest from the $O(n^2)$ different lines implicitly defined by a set of n points in the plane. By dualization this is reduced to the problem of finding the kth smallest x-coordinate of the vertices in an arrangement of n lines in the plane. A number of $O(n \log n)$ algorithms are known for this problem, with the paper by Cole et al [8] being the first to achieve this bound.

The first step in the reduction is to note that the segment sums $s(i,j)$ are implicitly defined by a set of $n+1$ prefix sums, $P[j] = \Sigma_{i=1}^{j} a_i$ and $P[0] = 0$. This is analogous to lines implicitly defined by pairs of a set of n given points in the plane. We shall see in the next few paragraphs how this analogy is exploited.

Now define two sets of lines: a set of horizontal lines $H = \{l_i : y = -P[i] | 0 \leq i \leq n\}$ and a set of 45^0-inclined lines: $I = \{l_{i+n+1} : y = x - P[i] | 0 \leq i \leq n\}$. Of the $O(n^2)$ intersections defined, the x-coordinates of the intersections of the lines $y = x - P[j]$ in I with the lines $y = -P[i]$ in H, where $i < j$, define all possible segment sums. These are called *feasible* intersection points in [13] and the problem is to select the kth smallest of these. This is not an exact reduction to the slope-selection problem, but this is only a minor obstacle as shown in [13].

Let $V_k : x = x_k$ be a vertical line through the kth feasible intersection point (x_k, y_k). We simulate a sorting algorithm (actually a parallel sorting algorithm like the AKS sorting network [2]) at x_k to determine the sorted order π of the lines of H and I with respect to the y-values of their intersections with V_k. When comparing an ordering π_1 at $x = c_1$ to an ordering π_2 at $x = c_2$, the inversion of the intersection order of a pair of lines l_i and l_j implies that these lines intersect between $x = c_1$ and $x = c_2$. An inversion is said to be *feasible* if the corresponding intersection point is feasible. In order to resolve a comparison between two lines l_i and l_j, we need an oracle that returns a count of the number of feasible intersections to the left of the line $x = x_{ij}$, where (x_{ij}, y_{ij}) is the intersection of the lines l_i and l_j. By comparing this count with k, we can decide which side of V_k contains (x_{ij}, y_{ij}). Knowing this determines the relative ordering of l_i and l_j at V_k.

Lee and Lin provide such a counting oracle whose complexity is in $O(n \log n)$. This oracle, used in

conjunction with a parallel sorting scheme and Cole's technique of slowing down a sorting network, solves the problem in $O(n \log^2 n)$ time. An approximate counting oracle is then used to whittle away a further $\log n$ factor.

The essence of this approximate counting oracle is to maintain the following invariant: $FI_{exact} \leq FI_{approx} \leq \alpha n$, where FI_{exact} and FI_{approx} are respectively a count of the exact number of feasible intersections and an approximate number of these in an interval $[x_l, x_r]$ that contains x_k, while α is a factor that decreases as the parallel sort progresses.

This is accomplished by maintaining an approximate count of the inversions at the boundaries of the interval $[x_l, x_r]$. It is a modification of Cole's idea [8]. At $x = x_l$ the lines in H are partitioned into a set of fixed-size blocks $B^l = \{b^l_1, b^l_2, \ldots, b^l_{\frac{n}{m_l}}\}$ such that for all $1 \leq i < j \leq \frac{n}{m_l}$, no line in b^l_i will intersect $x = x_l$ at a point higher than any line in b^l_j. A similar partition B^r is maintained for the end-point x_r. At x_l the actual position of the intersection of a line in I is accurate to within a block. That is if the actual intersection is inside a block b^l_i then in the approximate order it is assumed to lie between b^l_{i-1} and b^l_i (see Fig. 1). At x_r this intersection will be placed between b^r_i and b^r_{i+1} if it actually lies in the block b^r_i.

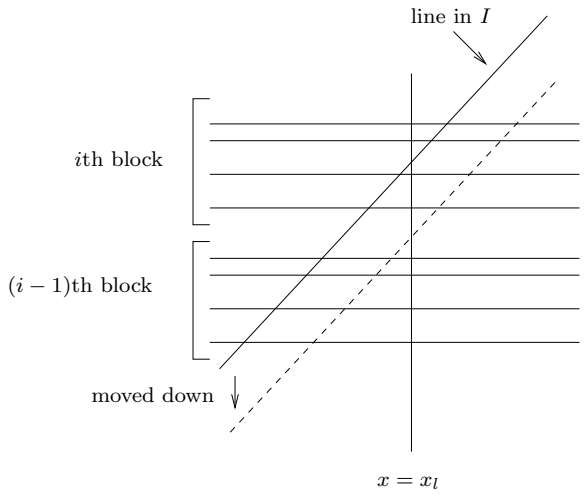

Figure 1: In the approximate ordering at $x = x_l$, a line in I is inserted between b^l_{i-1} and b^l_i

This means that the approximate inversion count at x_l is an underestimate by at most $m_l n$, and the count at x_r is an overestimate by at most $m_r n$. Whenever the oracle receives a query in the form of some vertical line $x = q$, it needs to determine the side of V_k that contains the query line. If $q < x_l$ or $q > x_r$, then the query line lies to the left or right, respectively, of V_k. Otherwise, we split $[x_l, x_r]$ at q, and keep the interval that contains x_k. As the algorithm progresses, the number of lines in each block is halved when necessary to maintain certain invariants regarding the accuracy of the approximate orderings.

When α reduces to a small constant value, the set of feasible intersections X inside the current interval $[x_l, x_r]$ is determined by an enumeration oracle, while the number of feasible intersection points k' to the left of the line $x = x_l$ is found by invoking an exact counting oracle, both due to Lin and Lee. Now a linear-time selection is used to find x_k, which is the point of X with the $(k - k')$th smallest x-coordinate. However, if the sorting algorithm finishes before α is small enough, then x_k is found using the sorted order of the lines at V_k. (The lines responsible for x_k will just have been inverted in the ordering π, and so x_k is the largest x-coordinate of the points of intersection of lines adjacent and inverted in π.)

5 Expander Graphs

Expander graphs are an extremely versatile tool. They have been applied to find a simplified proof of the celebrated PCP theorem [9], to find optimal codes [17], and to problems in computational Geometry [11].

Intuitively explained, an expander graph is a sparse graph that is well-connected. Formally speaking, an expander graph is a d-regular graph on a set V of n vertices such that for any subset $S \subset V$ of at most $n/2$ vertices, there are at least $c|S|$ vertices of $V - S$ connected to S. Such a d-regular graph is called an (n, d, c) expander graph. Explicit constructions of expander graphs have been given by Lubotzky, Phillips, and Sarnak (LPS, for short) [14] and, independently, by Margulis [15] under suitable restrictions on n and d. Katz and Sharir [10] make use of these graphs, with sets of lines as vertex sets, to avoid Cole's use of a sorting algorithm.

6 Alternate algorithm for the kth smallest segment sum

Let L be an arrangement of n lines in the plane. Suppose by some magic we could bunch all the intersection points into an ordered set of groups each of size αn^2 and locate the kth *feasible* intersection point in one of these groups. If in the process of location we can also determine the rank of the kth feasible intersection point within this group, we can repeat this process on this group of points of reduced size. Hopefully, by doing this $O(\log n)$ times we should be able to identify the kth feasible intersection point.

Katz and Sharir [10] showed how to use expander

graphs to do this when we are looking for the kth of all the intersection points. The salient feature of their algorithm is to maintain over $O(\log n)$ stages a trapezoidized vertical slab $s_j = [x_l, x_r]$ (at stage j) that contains the kth intersection point. Each trapezoid t_i in the slab s_j intersects a subset L_i^j of lines from L. The trapezoidation is such that

- each trapezoid is intersected by at most $(3/r)^j n$ lines, where r is a fixed large constant such that $9r^2 \leq d$
- the total count of $|L_i^j|$ over all trapezoids is at most $3n$

Expander graphs are constructed over each L_i^j. The edges of these graphs (corresponding to intersection points of the arrangement of L) are used to construct $O(r^2)$ slabs and $O(r)$ trapezoids per slab, for each t_i. The intersections of L with these new trapezoids are used to divide s_j into a constant number of new vertical slabs. Using a binary search on these new slabs, facilitated by Lin and Lee's approximate counting oracle, we can locate the slab s_{j+1}, that contains V_k, in a constant number of oracle queries. The trapezoidation of s_{j+1} is obtained from the $O(r^3)$ trapezoids we constructed from the expander graphs.

This process continues until each trapezoid in our slab intersects a constant number of lines. Finally, we need to enumerate the feasible intersection points inside our final slab if the kth feasible intersection point has not already been discovered. The trapezoidation of the slab provides us with such an enumeration. We determine the number of feasible intersection points inside each one of the trapezoids, by intersecting pairwise the lines that intersect the trapezoid. Since we need to know the rank of the kth feasible intersection point in this list, we invoke Lin and Lee's counting oracle to determine the number of feasible intersection points to the left of the left slab-boundary.

Each query to the approximate counting oracle takes $O(n)$ time to answer, and there are $O(\log n)$ queries. The total cost of maintaining the approximate counting oracle is in $O(n \log n)$. The total cost of the expander graph construction over all the stages is dominated by $O(n \log n)$. Hence the worst-case time complexity of this algorithm is in $O(n \log n)$.

7 Conclusion

The foremost open problem is to come up with a really simple deterministic $O(n \log n)$ algorithm for this problem. The other open problems are to explore the possibility of extending these techniques to higher dimensions. According to [10] the kth intersection point of an arrangement of hyperplanes in d dimensions can be found in time that is close to $O(n^{d-1})$. This gives cause for hope.

References

[1] A. Aggarwal, M. M. Klawe, S. Moran, P. W. Shor, and R. E. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2:195–208, 1987.

[2] M. Ajtai, J. Komlós, and E. Szemerédi. Sorting in c log n parallel steps. *Combinatorica*, 3(1):1–19, 1983.

[3] S. E. Bae and T. Takaoka. Improved algorithms for the k-maximum subarray problem. *Comput. J.*, 49(3):358–374, 2006.

[4] F. Bengtsson and J. Chen. Efficient algorithms for k maximum sums. *Algorithmica*, 46(1):27–41, 2006.

[5] J. Bentley. Programming pearls: algorithm design techniques. *Commun. ACM*, 27(9):865–873, 1984.

[6] C.-H. Cheng, K.-Y. Chen, W.-C. Tien, and K.-M. Chao. Improved algorithms for the k maximum-sums problems. *Theor. Comput. Sci.*, 362(1):162–170, 2006.

[7] R. Cole. Slowing down sorting networks to obtain faster sorting algorithms. *J. ACM*, 34(1):200–208, 1987.

[8] R. Cole, J. S. Salowe, W. L. Steiger, and E. Szemeredi. An optimal-time algorithm for slope selection. *SIAM J. Comput.*, 18(4):792–810, 1989.

[9] I. Dinur. The PCP theorem by gap amplification. *Electronic Colloquium on Computational Complexity (ECCC)*, 12(046), 2005.

[10] M. J. Katz and M. Sharir. Optimal slope selection via expanders. *Inf. Process. Lett.*, 47(3):115–122, 1993.

[11] M. J. Katz and M. Sharir. An expander-based approach to geometric optimization. *SIAM J. Comput*, 26(5):1384–1408, 1997.

[12] S. K. Kim. Linear-time algorithm for finding a maximum-density segment of a sequence. *Information Processing Letters*, 86(6):339–342, June 2003.

[13] T.-C. Lin and D. T. Lee. Efficient algorithms for the sum selection problem and k maximum sums problem. In S. K. Madria, K. T. Claypool, R. Kannan, P. Uppuluri, and M. M. Gore, editors, *ISAAC*, volume 4317 of *Lecture Notes in Computer Science*, pages 460–473. Springer, 2006.

[14] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.

[15] G.A. Margulis. Explicit group-theoretical constructions of combinatorial schemes and their application to the design of expanders and concentrators. *PINFTRANS: Problems of Information Transmission (translated from Problemy Peredachi Informatsii (Russian))*, 24:39–46, 1988.

[16] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. ACM*, 30(4):852–865, 1983.

[17] M. Sipser and D. A. Spielman. Expander codes. *IEEE Transactions on Information Theory*, 42(6):1710–1722, 1996.