Algorithms for the Maximum Subarray Problem Based on Matrix Multiplication

Hisao TAMAKI* and Takeshi TOKUYAMA**

*Department of Computer Science, Faculty of Science and Engineering, Meiji University,
Kawasaki, Kanagawa, 214, Japan
E-mail: tamaki@cs.meiji.ac.jp

**Graduate School of Information Sciences, Tohoku University,
Sendai, 980-8577, Japan
E-mail: tokuyama@dais.is.tohoku.ac.jp

Received April 28, 2000; final version accepted November 18, 2000

Given an $M \times N$ array of reals, we want to find a rectangular contiguous subarray such that the sum of the entries in the subarray is maximized. Since Bentley posed this problem in his Programming Pearls column in 1984 with an $O(NM^2)$ time solution, no progress on the sequential complexity has been reported to date. We give the first subcubic algorithm, by reducing the problem to “funny matrix multiplication”, where the scalar product and addition in usual matrix multiplication are replaced by addition and max operations, respectively. We also give a faster ϵ -approximation algorithm via the same reduction.

KEYWORDS: Theoretical Computer Science, Algorithms, Complexity, Maximum Subarray, Matrix Multiplication

1 Introduction

In his “Programming Pearls” column of September 1984 issue of CACM [4], J. Bentley gave the following *maximum subarray problem* as a research problem.

“We are given an $M \times N$ array of reals, and we must find the maximum sum contained in any rectangular subarray. What is the complexity of this problem?”

In the array of Figure 1, the subarray bounded by the bold rectangle gives the maximum subarray, which has an array sum of 16.

We will assume without loss of generality that $M \leq N$. A naive solution checks all of $O(M^2N^2)$ rectangles, which needs $O(M^2N^2)$ (or more) time. However, it is not difficult to design an $O(NM^2)$ time algorithm for this problem. The problem is a two-dimensional version of the *maximum subinterval problem*, which computes the maximum sum contained in any subinterval of a given list of length N of real numbers. In the article [4], Bentley discussed history and algorithms on the maximum subinterval problem, and finally reported a dynamic programming algorithm designed by Kadane [4] attaining the optimal linear time bound.

If we consider $M(M + 1)/2$ column intervals that determine the width of the rectangle, the maximum subarray problem generates $M(M + 1)/2$ maximum subinterval problems on the row index. Therefore, applying Kadane’s algorithm for each of these $M(M + 1)/2$ problems, we can be solved the maximum subarray problem in $O(NM^2)$ time. Bentley reported this algorithm in the solution section of his column of November 1984 issue [5]. Also, $O(\log N)$ time CREW parallel algorithm using $NM^2/\log N$ processors is known [18]. There are several possible heuristic methods to compute the exact or an approximate solution efficiently if the distribution of

-2	4	0	-5	-2	2	-3	2
4	-2	-1	3	2	1	-1	2
-5	6	3	-5	-1	3	-2	-1
-1	-1	3	-1	4	1	1	0
3	-3	2	0	3	-3	-2	-2
-2	1	-2	1	-1	3	-1	1
2	-4	0	1	0	-3	-1	2
1	1	-2	-1	1	2	1	-2

Fig. 1 Maximum subarray problem.

values in the input array has a good property; Indeed, Fisher et al. [9] study special cases and also some modified problems.

However, to the authors' knowledge, no improvement on the $O(NM^2)$ upper bound has been done so far. On the negative side, only $\Omega(MN)$ trivial lower bound is known. Also, little theoretical study has been done on approximation algorithms with guaranteed performance.

In this paper, we first observe that a simple application of the divide-and-conquer strategy leads to a $(1/2)$ -approximation algorithm, i.e., an algorithm that finds a subarray with its sum at least half the value of the optimal solution, whose running time is $O(NM \log M)$.

Then, we apply the divide-and-conquer strategy again to a subproblem that arises from this approach. This nested application of the divide-and-conquer strategy reduces the maximum subarray problem to the problem of computing the "funny matrix multiplication" of $M \times N$ and $N \times M$ matrices.

Given $m \times n$ and $n \times l$ real matrices $A = (a_{ij})$ and $B = (b_{ij})$, the funny matrix product $A \odot B$ is the $m \times l$ matrix $C = (c_{ij})$ with $c_{ij} = \max_{1 \leq k \leq n} (a_{ik} + b_{kj})$. Thus, funny matrix multiplication is matrix multiplication on the semi-ring where the scalar addition is max and the scalar multiplication is $+$. Funny matrix multiplication is well-studied, since its computational complexity is known to be equivalent to that of all-pairs shortest paths in weighted directed graphs (See [1]). In a computation model in which we can do any infinite precision arithmetic computation in $O(1)$ step, it is known that a funny matrix multiplication of two $N \times N$ matrices can be done in the same complexity as the usual matrix multiplication, that is, in time $O(N^\omega)$ where ω is 2.376 in the current best result [6]. However, this model is generally considered too unrealistic. On the other hand, a cubic lower bound is known under a straight-line computation model which only permit plus and maximum as basic operations [14]. Fredman [10] shows that, for each N , there "exists" an algorithm that computes the funny matrix product of two $N \times N$ matrices in $O(N^{5/2})$ time. Unfortunately, the construction and storage of such an algorithm, given N , would require time and space exponential in N . Applying this algorithm to submatrices of roughly logarithmic size, Fredman constructs a subcubic algorithm with running time $O(N^3(\log \log N)^{1/3}/(\log N)^{1/3})$ in a standard computation model which allow polynomial-precision arithmetic including subtraction and direct access to a polynomial size memory; this is subsequently improved to $O(N^3(\log \log N)^{1/2}/(\log N)^{1/2})$ by Takaoka [19]. Another direction is taken by Alon, Galil, and Margalit [3] who observe, in their work on all-pairs shortest path problems, that if each entry of the matrices is an integer with its absolute value bounded by Q then funny matrix multiplication can be done in $O(N^\omega Q \log Q)$ time, using the fast algorithm for usual matrix multiplication.

Our reduction of the maximum subarray problem to funny matrix multiplication enables us to borrow these accumulated results. In particular, we have a subcubic algorithm owing to the Fredman's result, and a much faster ϵ -approximation algorithm using the above observation of Alon, Galil, and Margalit. In the following theorem, an ϵ -approximate solution is the solution whose value is at least $(1 - \epsilon)$ times the optimal value. We use a convention that $\log(N/M) = O(1)$ if $N = M$.

Theorem 1.1 *The maximum subarray problem can be solved in $O(NM^2(\log \log M/\log M)^{1/2} \log(N/M))$ time. Moreover, an ϵ -approximate solution of the maximum subarray problem can be computed in $O(\epsilon^{-1}(-\log \epsilon)NM^{\omega-1} \log(N/M))$ time.*

Besides its pedagogical value, the maximum subarray problem has several practical applications. The original application referred to by Bentley (page 868, [4]) is a problem arising in pattern recognition: the array corresponds to the brightness of pixels in a pixel grid, normalized so that the average brightness is 0 (or some given number), and the maximum subarray is the rectangle with highest total brightness.

Recently, another application in database mining is gaining attention [2,11]. Indeed, our interest in the problem has stemmed from our work on extracting so called association rules from a large database. Consider a pair of numeric attributes, say, *Age*() and *Income*(), in an customer database of a store. Numeric attributes are usually discretized into buckets, hence the data corresponding to *Age*() and *Income*() becomes an $M \times N$ array. Let $g(x)$ be a function on the database records, for example, the profit from the customer x (hence, possibly negative). In database query, the user can input $(\text{Age}, \text{Income})$ and a grid region R , and obtain $g(R)$, which is the sum of $g(x)$ over all records satisfying $(\text{Age}(x), \text{Income}(x)) \in R$, using *range query*. Instead of asking users to input R explicitly, it is useful that the system answers the grid region R in a given family of regions maximizing $g(R)$ automatically. This operation plays a basic role in the parametric optimization algorithms for computing optimized association rules with respect to several criteria in [11,12,16,22]. The family of grid rectangles is a popular choice, because it is intuitive and easy to include in SQL framework, and thus, we encounter the problem of finding the maximum subarray.

The maximum subarray problem also has a potential application in computing discrepancy. The discrepancy of a point set P in a unit square with respect to a family χ of regions in the unit square is $\max_{R \in \chi} |\mu(R) - \mu_P(R)|$, where $\mu(R)$ is a measure of R , and $\mu_P(R) = |R \cap P|/|P|$. Discrepancy with respect to the family of isothetic rectangles is a very important tool to evaluate quality of sample point generation methods in Monte Carlo simulation, and several generating methods of low-discrepancy sequences are known [17,20,21]. Computation of discrepancy of a given point set is studied in [7,8]. If we restrict the family of regions to grid rectangles,

the computation of discrepancy is reduced to the maximum subarray problem.

2 Maximum subarray problem

We are given an $M \times N$ array $A = (a_{i,j})$ of reals. The first index is called the row index, and the second is called the column index. Without loss of generality, we assume that $M \leq N$. A rectangular subarray $R = I_R \times J_R$ of A is a direct product of closed intervals I_R and J_R of row and column indices.

Let $S(R) = \sum_{(i,j) \in R} a_{i,j} = \sum_{i \in I_R} \sum_{j \in J_R} a_{i,j}$. Our objective is to find R_{opt} maximizing $S(R)$. We assume that $S(R_{opt})$ is positive; Otherwise, all the entries are non-positive, and the problem is reduced to find the maximum entry of A , and can be solved trivially. We say a subarray R_{approx} is an ϵ -approximate solution of the maximum subarray problem if $S(R_{approx}) \geq (1 - \epsilon)S(R_{opt})$.

From now on, without loss of generality, we assume that $M = 2m$ and $N = 2n$ are even numbers. We regard the array as an array of pixels (squares) formed by $M + 1$ horizontal and $N + 1$ vertical grid lines. The center-most horizontal (resp. vertical) grid line is called the *center row separator* (resp. *center column separator*). The intersection of the center row separator and the center column separator is called the *center grid point*. A rectangular subarray R is called *row-central* (resp. *column-central*) if its interior intersects the center row (resp. column) separator. R is called *central* if it is both row-central and column-central.

In our divide-and-conquer approach, we will consider some variants of the maximum subarray problem, where the following assumptions are imposed. We say that a given array satisfies the *centrality assumption* if every maximum subarray of the array is central. Similarly, the *row-centrality* (resp. *column-centrality*) assumption states that every maximum subarray is row- (resp. column-) central. The *central* (resp. *row-central*, *column-central*) *maximum subarray problem* is to solve the maximum subarray problem under the centrality (resp. row-centrality, column-centrality) assumption.

2.1 A (1/2)-approximation algorithm

As a warm-up, we give a simple (1/2)-approximation algorithm with time complexity $O(NM \log M)$. We first deal with the row-central maximum subarray problem. We consider the family of subarrays each of whose boundary rectangle contains a segment on the center row separator, and compute the maximum subarray R_{half} in this family. The maximum row-central subarray is a union of two rectangles in the above family, and hence the sum in the subarray cannot exceed $2S(R_{half})$. Hence, R_{half} is a (1/2)-approximate solution of the row-central maximum subarray problem.

Without loss of generality, we assume that R_{half} lies above the center row separator. Then, $R_{half} = [k, m] \times [i, j]$, attaining

$$\max_{1 \leq k \leq m} [\max_{1 \leq i' \leq j' \leq N} \{S([k, m] \times [i', j'])\}].$$

For each k , we construct a list $A_{[k,m]}(\cdot)$ defined by $A_{[k,m]}(t) = \sum_{s \in [k,m]} a_{s,t}$ for $1 \leq t \leq N$ from the original two-dimensional array A . The computation of lists $A_{[k,m]}(\cdot)$ for $k = 1, 2, \dots, m$ can be done in $O(MN)$ time, by using a simple decremental strategy with respect to k . $S([k, m] \times [i, j]) = \sum_{t \in [i, j]} A_{[k,m]}(t)$, which is the sum of entries in the interval (i.e. sublist) $[i, j]$ of $A_{[k,m]}(\cdot)$. The interval $[i, j]$ maximizing this sum can be computed in $O(N)$ time by using Kadane's algorithm. Since there are $m = M/2$ lists, the time complexity is $O(MN)$ in total. Hence, the rectangle R_{half} can be computed in $O(MN)$ time.

The (1/2)-approximation algorithm of the maximum subarray problem divides the array with the center column separator, computes the (1/2)-approximate solution in each halfplane recursively, and outputs the maximum among these two recursive solution and the (1/2)-approximate solution of the column-central maximum subarray problem. It is easy to see that this gives (1/2)-approximation, and the running time is $O(MN \log M)$.

2.2 Nested divide-and-conquer algorithm

In order to obtain time complexities given in our main theorem (Theorem 1.1), we use the following nested divide-and-conquer strategy.

Let $F(M, N)$, $F_{row}(M, N)$, $F_{column}(M, N)$, and $F_{center}(M, N)$ be the time needed to compute the maximum subarray, row-central maximum subarray, column-central maximum subarray, and central maximum subarray problems, respectively. Since one-dimensional problem (maximum subinterval problem) can be solved in linear time [4], $F(M, 1) = O(M)$, $F_{row}(M, 1) = O(M)$, $F(1, N) = O(N)$, and $F_{column}(1, N) = O(N)$. Because of symmetry, we have

$$F_{column}(M, N) = F_{row}(N, M). \quad (1)$$

We subdivide the grid array into NW, SW, NE and SE parts with the center row separator and center column separator. Then, R_{opt} is either contained in one of these four parts, column central or row central. Hence, we have the recursion

$$F(M, N) \leq F_{\text{row}}(M, N) + F_{\text{column}}(M, N) + 4F(M/2, N/2). \quad (2)$$

For computing the row-central (resp. column-central) maximum subarray, we subdivide the grid with the center column (resp. row) separator. The optimal rectangle is either central or contained in one of the halfplanes divided by the center column (resp. row). Hence, we have the following two recursions:

$$F_{\text{row}}(M, N) \leq F_{\text{center}}(M, N) + 2F_{\text{row}}(M, N/2). \quad (3)$$

$$F_{\text{column}}(M, N) \leq F_{\text{center}}(M, N) + 2F_{\text{column}}(M/2, N). \quad (4)$$

The following is the key lemma, to which we will give a proof in the subsequent sections.

Lemma 2.1 *The central maximum subarray can be computed in $O(NM^2(\log \log M/\log M)^{1/2})$ time. An ϵ -approximate solution of the central maximum subarray problem can be computed in $O(\epsilon^{-1}(-\log \epsilon)NM^{\omega-1})$ time.*

Combining the above key lemma with formulae 3, 4, and 1, we have the following corollary, from which we can obtain our main theorem by using formula 2.

Corollary 2.2 *The row-central and column-central maximum subarrays can be computed in $O(NM^2(\log \log M/\log M)^{1/2} \log(N/M))$ time and $O(NM^2(\log \log M/\log M)^{1/2})$ time, respectively. ϵ -approximate solutions of the row-central and column-central maximum subarray problems can be computed in $O(\epsilon^{-1}(-\log \epsilon)NM^{\omega-1} \log(N/M))$ time and $O(\epsilon^{-1}(-\log \epsilon)NM^{\omega-1})$ time, respectively.*

3 Reducing the central subarray problem to funny matrix multiplication

3.1 Subcubic exact algorithms

We mainly consider the central maximum subarray problem in this section.

For $1 \leq i \leq m$ and $1 \leq j \leq n$, let $R_{i,j}^{NW}$, $R_{i,j}^{NE}$, $R_{i,j}^{SW}$, and $R_{i,j}^{SE}$ be the rectangular arrays defined by $[i, m] \times [j, n]$, $[m+1, m+i] \times [j, n]$, $[i, m] \times [n+1, n+j]$, and $[m+1, m+i] \times [n+1, n+j]$, respectively.

Let $b_{i,j}^{NW} = \sum_{(s,t) \in R_{i,j}^{NW}} a_{s,t}$, which is the sum of array entries in $R_{i,j}^{NW}$. Let B^{NW} be the matrix $(b_{i,j}^{NW})_{i=0,1,\dots,m-1; j=0,1,\dots,n-1}$. We analogously define B^{NE} , B^{SW} , and B^{SE} .

The matrices generated from the array of Figure 1 is illustrated in Figure 2, so that the north-west part of the figure shows B^{NW} , etc. For the figure, $S(R_{\text{opt}}) = 16$ is obtained as the sum of four circled entries.

We can compute these matrices in $O(NM)$ time easily. For matrices X and Y with a same shape, we define the sum $X + Y$ as usual, i.e. its (i, j) -th entry is $x_{i,j} + y_{i,j}$. tX denotes the transposition of a matrix X .

Lemma 3.1 *Let $X_{\text{left}} = B^{NW} \odot {}^tB^{SW}$, $X_{\text{right}} = B^{NE} \odot {}^tB^{SE}$, and $X = X_{\text{left}} + X_{\text{right}}$. Then, the maximum entry of X gives $S(R_{\text{opt}})$ for the central maximum array problem.*

Proof: By definition, $S(R_{\text{opt}}) = \max_{1 \leq i,j,k,l} S([i, m+j] \times [k, n+l])$. Partitioning it into NW, SW, NE, and SE parts, $S(R_{\text{opt}}) = \max_{i,j,k,l} \{S([i, m] \times [k, n]) + S([m+1, m+j] \times [k, n]) + S([i, m] \times [n+1, n+l]) + S([m+1, m+j] \times [n+1, n+l])\}$. If we take the maximum with respect to column indices first, we have $S(R_{\text{opt}}) = \max_{i,j} \{S([i, m] \times [k, n]) + S([m+1, m+j] \times [k, n])\} + \max_l \{S([i, m] \times [n+1, n+l]) + S([m+1, m+j] \times [n+1, n+l])\}$. Since the i, j entries of $X_{\text{left}} = B^{NW} \odot {}^tB^{SW}$ and $X_{\text{right}} = B^{NE} \odot {}^tB^{SE}$ equal $\max_k \{S([i, m] \times [k, n]) + S([m+1, m+j] \times [k, n])\}$, and $\max_l \{S([i, m] \times [n+1, n+l]) + S([m+1, m+j] \times [n+1, n+l])\}$, respectively, we have that $S(R_{\text{opt}})$ is the maximum entry of X . $\square$

Fig. 2 Matrices.

Once we know the maximum entry of X is $x_{i,j}$, $R_{opt} = [i, m + j] \times J$ for some interval J . The interval J can be computed in $O(N)$ time by applying Kadane's algorithm for the maximum subinterval problem. Hence, the time complexity $F_{center}(M, N)$ is $O(NM + f(M/2, N/2))$, where $f(M, N)$ is the time for computing funny matrix multiplication of an $M \times N$ matrix and an $N \times M$ matrix.

The funny matrix multiplication algorithm of Fredman/Takaoka cited in the introduction, i.e., the $O(N^3(\log \log N / \log N)^{1/2})$ time algorithm for two $N \times N$ matrices, implies that $f(M, N) = O(NM^2(\log \log M / \log M)^{1/2})$. Thus, the central maximum subarray problem can be solved in the same time complexity, establishing the first part of Lemma 2.1.

It seems difficult to design an exact algorithm with exponent less than 3 for the maximum subarray problem with this approach. Therefore, it is useful to design an efficient approximation algorithms.

3.2 ϵ -approximation algorithm

Let K be the maximum among the entries of matrices B^{NW} , B^{NE} , B^{SW} , and B^{SE} . It is easy to see that $S(R_{opt}) \leq K$, and we assume that $K > 0$; otherwise the optimal solution of the original maximum subarray problem cannot be central. Let $Q = \lceil 1/\epsilon \rceil$.

First, we replace each entry less than $-3K$ in B^{NW} , B^{NE} , B^{SW} , and B^{SE} with $-\infty$. We next multiply $4Q/K$ to each entry, and round to its floor integer. The modified matrices are denoted by $\tilde{B}^{NW}$, $\tilde{B}^{NE}$, $\tilde{B}^{SW}$, and $\tilde{B}^{SE}$. These matrices are integral, and the maximum absolute value of entries is $12Q$, except $-\infty$. Let R_{approx} be the rectangle corresponding to the maximum entry of

$$X_{approx} = (\tilde{B}^{NW} \odot {}^t\tilde{B}^{NE}) + (\tilde{B}^{SW} \odot {}^t\tilde{B}^{SE}).$$

The following lemma certifies that R_{approx} gives an ϵ -approximate solution to the central maximum subarray.

Lemma 3.2 $S(R_{opt}) - S(R_{approx})$ is at most ϵK if R_{opt} is central.

Proof: If the maximum entry of X is negative, the maximum subarray is not central, and we need to do nothing. If it is positive, none of four additive terms contributing to that entry is less than $-3K$, and the corresponding entry of X_{approx} is at least $(4Q/K)S(R_{opt}) - 4$, since the rounding loss is at most 4. The maximum entry $x_{max} = \tilde{b}^{NW} + \tilde{b}^{SW} + \tilde{b}^{NE} + \tilde{b}^{SE}$ (we omit subscript indices) of X_{approx} is at most $(4Q/K)S(R_{approx})$. Hence, $(4Q/K)S(R_{approx}) \geq (4Q/K)S(R_{opt}) - 4$, and thus, $S(R_{opt}) - S(R_{approx}) < K/Q \leq \epsilon K$. $\square$

Hence, it suffices to compute the maximum entry of X_{approx} . The following lemma is given in Alon et al. [3]. They consider min instead of max, and thus, we give an outline of the proof.

Lemma 3.3 Given a pair of $N \times N$ integral matrices whose maximum absolute value of the entries is bounded by Q' except $-\infty$, their funny matrix multiplication can be computed in $O(Q'(\log Q')N^\omega)$ time.

Proof: We replace each $-\infty$ entry by $-4Q'$. For a $N \times N$ matrix $C = (c_{i,j})$ and $D = (d_{i,j})$, we define the matrix $\Psi(C)$ and $\Psi(D)$, which has $(N+1)^{Q'-c_{i,j}}$ and $(N+1)^{Q'-d_{i,j}}$ as their (i,j) entries. Compute the usual product $Y = \Psi(C) \cdot \Psi(D)$ of $\Psi(C)$ and $\Psi(D)$. The entry $y_{i,j}$ of Y is $\sum_{k=1}^N (N+1)^{2Q'-c_{i,k}-d_{k,j}}$, and it is divisible by $(N+1)^s$ iff for all k , $2Q' - c_{i,k} - d_{k,j} \geq s$. Hence, we find the largest such s by using binary search, and if $2Q' - s$ is not smaller than $-2Q'$, it gives $\max_k (c_{i,k} + d_{k,j})$; otherwise, $\max_k (c_{i,k} + d_{k,j}) = -\infty$. For the time complexity, see [3]. $\square$

Thus, applying the above lemma for $Q' = 12Q$, we can compute $\tilde{B}^{NW} \odot {}^t\tilde{B}^{NE}$ and $\tilde{B}^{SW} \odot {}^t\tilde{B}^{SE}$, in $O(\epsilon^{-1}(\log \epsilon^{-1})NM^{\omega-1})$ time, and hence, obtain the maximum entry of X_{approx} in additional $O(NM)$ time. This completes the proof of Lemma 2.1, and hence, the main theorem.

Note: If there are at most $T = o(M^2)$ entries in X_{approx} whose value is more than $(1 - \epsilon)x_{max}$, where x_{max} is the maximum value of X_{approx} , we can re-compute only these entries exactly using $O(NT)$ time, and transform the approximate algorithm into an exact one. Unfortunately, if we consider a chess board, and put $1 + \delta(p)$ for each black pixel p and -1 for each white pixel, where $\delta(p)$ is a small noise, there may be $\Omega(M^2)$ entries in X_{approx} each of which is larger than $(1 - \epsilon)x_{max}$.

4 Concluding remarks

(1): An efficient approximation algorithm solving higher dimensional maximum subarray problem would be very useful in application to both data mining and Monte-Carlo simulation. In data mining, the dimension is practically low (say, less than 4); On the other hand, in Monte-Carlo simulation, it may happen that discrepancy in a high dimensional space concerns. We only have a naive extension to give an $O(\epsilon^{-1}(-\log \epsilon)N^{2d-4+\omega})$ time algorithm for the $N \times \dots \times N$ d -dimensional array, and its improvement is an interesting open problem.

(2): In [11], the notion of maximum gain region rule is extended to other families of two-dimensional grid regions, such as connected x -monotone regions and rectilinear convex regions. Indeed, the maximum gain region rule can be computed in the optimal $O(MN)$ time for the connected x -monotone regions. Strangely enough, the computation of an optimized solution in the family of connected x -monotone regions, which has exponential number of regions, is easier than the family of rectangles. On the other hand, if we consider the maximum square array, there are only $O(NM^2)$ candidate squares. However, the matrix multiplication method

given in this paper does not work, and it remains open to improve the almost trivial $O(NM^2)$ bound, even if we permit approximation algorithms.

(3): The funny matrix multiplication is an all-pairs shortest paths problem on a weighted graph G . Average case performance of all-pairs shortest path problems is well-studied [13,15]; However, the edge weights of the network generated from instances of the maximum subarray problem do not have a typical uniform distribution, even if the original array entries have good distributions. It is desirable to study on the average performance of algorithms for computing the maximum subarray, and its d -dimensional extension.

REFERENCES

- [1] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms* Addison-Wesley, 1974.
- [2] R. Agrawal, T. Imielinski and A. Swami, Mining Association Rules between Sets of Items in Large Databases, *Proc. ACM SIGMOD Conf. on Management of Data*, (1993), 207–216.
- [3] N. Alon, Z. Galil and O. Margalit, On the Exponent of the All Pairs Shortest Path Problem, *Proc. 32nd FOCS* (1991), 569–575.
- [4] J. Bentley, Programming Pearls—Algorithm Design Techniques, *CACM* **27-9**, (1984), pp. 865–871.
- [5] J. Bentley, Programming Pearls—Perspective on Performance, *CACM* **27-11**, (1984), pp. 1087–1092.
- [6] D. Coppersmith and S. Winograd, Matrix Multiplication via Arithmetic Progression, *J. Symbolic Computation* **9** (1990), pp. 251–280.
- [7] D. Dobkin and D. Eppstein, Computing the Discrepancy, *Proc. 9th ACM Symp. Computational Geometry* (1993), pp. 47–52.
- [8] D. Dobkin and D. Gunopulos, Computing the Rectangle Discrepancy, *Proc. 10th ACM Symp. Computational Geometry* (1994), pp. 385–386.
- [9] P. Fisher, K. U. Höffgen, H. Lefmann and T. Luczak, Approximations with Axis Parallel Rectangles, *Proc. FCT'93* Springer LNCS 710, (1993), pp. 244–255.
- [10] M. L. Fredman, New Bounds on the Complexity of the Shortest Path Problem *SIAM J. Comput.* **5** (1976), pp. 83–89.
- [11] T. Fukuda, Y. Morimoto, S. Morishita and T. Tokuyama, Data Mining Using Two-Dimensional Optimized Association Rules: Scheme, Algorithms, and Visualization, *Proc. ACM SIGMOD Conf. on Management of Data* (1996), pp. 13–23.
- [12] T. Fukuda, Y. Morimoto, S. Morishita and T. Tokuyama, Constructing Efficient Decision Trees by Using Optimized Association Rules, *Proc. 22nd VLDB Conference* (1996), pp. 146–155.
- [13] D. Kerger, D. Koller and S. J. Phillips, Finding the Hidden Path; Time bounds for All-Pairs Shortest Path, *SIAM J. Comput.*, **22** (1993), pp. 1199–1217.
- [14] L. R. Kerr, *The effect of algebraic structure on the computational complexity of matrix multiplications*, Ph. D. Thesis, Cornell University, 1970.
- [15] K. Mehlhorn and B. Pribe, On the All-Pairs Shortest-Path Algorithm of Moffat and Takaoka, *Random Structures and Algorithms* **10** (1997), pp. 205–220.
- [16] Y. Morimoto, T. Fukuda, S. Morishita and T. Tokuyama, Implementation and Evaluation of Decision Trees with Range and Region Splitting, *Constraints*, **2** (1997), pp. 401–427.
- [17] H. Neiderreiter, *Random Number Generation and Quasi-Monte Carlo Methods*, CGMS-NSF Conference Series in Appl. Math. 63, SIAM, 1992.
- [18] K. Perumalla and N. Deo, Parallel algorithms for maximum subsequence and maximum subarray, *Parallel Processing Letters* **5-3**, (1995), pp. 367–73.
- [19] T. Takaoka, A new upper bound on the complexity of the all pairs shortest path problem, *IPL* **43** (1992), pp. 195–199.
- [20] S. Tezuka, *Uniform Random Numbers; Theory and Practice*, Kluwer Academic Publishers, 1995.
- [21] S. Tezuka and T. Tokuyama, A Note on Polynomial Arithmetic Analogue of Halton Sequences, *Acm Trans. Modeling and Comput. Simulation* **4**, (1994), pp. 279–284.
- [22] K. Yoda, T. Fukuda, Y. Morimoto, S. Morishita and T. Tokuyama, Computing Optimized Rectilinear Regions for Association Rules and Highly Accurate Decision Trees, *Proc. 3rd International Conference on Knowledge Discovery and Data Mining*, pp. 96–103, 1997.