DOI: 10.5604/01.3001.0010.7507

MAXIMUM SUBARRAY PROBLEM OPTIMIZATION FOR SPECIFIC DATA

Tomasz Rojek
Cracow University of Technology, Faculty of Mechanical Engineering, Institute of Applied Informatics

Abstract. *The maximum subarray problem (MSP) is to the find maximum contiguous sum in an array. This paper describes a method of Kadanes algorithm (the state of the art) optimization for specific data (continuous sequences of zeros or negative real numbers). When the data are unfavourable, the modification of the algorithm causes a non significant performance loss (1% > decrease in performance). The modification does not improve time complexity but reduces the number of elementary operations. Various experimental data sets have been used to evaluate possible time efficiency improvement. For the most favourable data sets an increase in efficiency of 25% can be achieved.*

Keywords: Algorithm design and analysis, maximum subarray problem, Kadane's algorithm, optimization

OPTYMALIZACJA PROBLEMU NAJWIĘKSZEJ PODTABLICY DLA SPECYFICZNYCH DANYCH

Streszczenie*. Problem najwiekszej podtablicy to inaczej znalezienie podciągu, którego suma na największą wartość. Artykuł opisuje optymalizację algorytmu Kadane dla specyficznych danych (z powtarzającymi się ciągami zer lub liczb negatywnych). W przypadku niekorzystnych danych wejściowych zaproponowa modyfikacja nieznacznie spowalnia działanie algorytmu (mniej niż 1% szybkości działania). Ulepszenie algorytmu nie zmienia rzędu asymptotycznego tempa wzrostu, lecz zmniejsza ilość elementarnych operacji. Eksperymenty wykazały, że dla sprzyjających danych możemy zmniejszyć efektywny czas działania algorytmu o 25%.*

Słowa kluczowe: analiza i projektowanie algorytmów, problem maksymalnej podtablicy, algorytm Kadane, optymalizacja

Introduction

The maximum subarray problem was described by U. Grenander at Brown Univer sity in 1977 during his study on the pattern recognition field. There are numerous algorithms which solves the problem in range from brute force method with time complexity $O(n^2)$ to Kadane's algorithm with time complexity O(n). Maximum subarray problem solutions are used in data mining [10], pattern recognition [5, 9], biological sequence analysis [6, 1, 11], computer vision [2] etc.

The maximum subsequence problem for a one-dimensional array is to find a contiguous subarray which has the largest sum of elements. At least one element of the array should be a positive real number. If the array elements are all non-negative, the solution is the whole array. When all the numbers are negative then the maximum subarray is the empty array.

Example. We are given a zero-indexed array X, such that:

$$X = [\ 5,4,1,\ 6,3,1,5,\ 5,6].$$

Maximum subarray sum is 10, the beginning of this subarray is on index 4, and the end is on index 8.

Kadane's algorithm [3] finds maximum contiguous subsequence in one-dimensional array. The input is an array X consisting of N real numbers. The output is the maximum sum found in any contiguous subarray of the input array. Its running time is linear (O(n)) The algorithm uses the dynamic programming approach. Kadane's algorithm scans the whole array from left (X[0]) to right (X[N-1]) and keeps track of the maximum sum subarray seen so far. Its pseudocode is given below.

ALGORITHM 1: Kadane's algorithm with complexity

```
Input: array - consisting of N real numbers
Output: maxSum - maximum contiguous sum
currentMax, maxSum = array[0] ;                          // 2
for i ← 1 to n ;                                  // 1 + n + (n − 1)
do
    if currentMax < 0 ;                                  // n − 1
    then
        currentMax := array[i] ;                         // (n−1)/2
    else
        currentMax+ = array[i] ;                         // (n−1)/2
    end
    if currentMax >= maxSum ;                            // n − 1
    then
        currentSum := currentMax ;                       // n − 1
    end
end
return maxSum;
```

On the right side of pseudocode are numbers (after //) represent the number of elementary operations which are done to execute certain line of code.

Below we describe in detail Kadane's algorithm analysis. The runtime complexity of Kadane's algorithm is O(n). This description focuses on number of elementary operations not asymptotic notation. Elementary operation is defined as one of the arithmetic operations (addition, subtraction, multiplication, division), comparisons (between two real numbers), assignments [7]. Let T (n) be the number of units of time taken for any input array size of n, required to solve the maximum subarray problem. For simplicity's sake, one can assume that all elementary operations take the same time. The analysis is given for the worst-case running time. Second line (*for i ← 1 to n*) *takes 1 + n + (n − 1)* due to: 1 for assignment operations (*i = 1*), n for checking condition (*i < n*) and (*n – 1*) for incrementing i. According to the above pseudocode one has:

$$T(n) = 2+1+n+(n-1)+(n-1)+(\frac{n-1}{2})+(\frac{n-1}{2})+$$
$$+(n-1)+(n-1)$$
$$T(n) = 6n-2$$

1. Materials and methods

In this section we describe improvement of Kadane's algorithm and explain for what kind of data is can be effectively used. Then we present test results for random generated data. Finally we demonstrate how much faster can be modified algorithm during searching specific regions on images.

1.1. Algorithm optimizing

Idea of optimizing Kadane's algorithm for specific data is based on the observation that the searched interval (containing solution) always starts and ends with a positive real number. All numbers before(after) first(last) positive should be negative or equal to zero.

For example, for a given zero-indexed array B consisting of N=11 real numbers:

$$B = [\ 2,-3,0,\ 1,5,2,-1,7,-8,1,0].$$

It can be noticed that the array containing the solution is between 4 (index) and 9 because all real numbers with an index less than 4 and greater than 8 are negative or equal to 0.

We can exclude part of the array which we know that there is no solution. In order to eliminate further calculations, we can searching for the first positive real number from left/right side.

To find the interval with the solution, the algorithm scans the whole array from the left until it finds real number x, where x > 0 and returns its index. This operation works analogously for the other side of the input array. The returned indices determine the range in which the array will be analyzed by the classic Kadane's algorithm. The pseudocode of improved version of the algorithm is shown below.

ALGORITHM 2: Improved Kadane's algorithm with complexity

```
Input: array - consisting of N real numbers
Output: maxSum - maximum contiguous sum
currentMax, maxSum = array[0]; begin = 0, end = n ;     // 2 + 2
for i ← 1 to n ;                                        // (1+n+(n−1))/2
do
    if array[i] > 0 ;                                   // (n−1)/2
    then
        begin = i ;                                     // 1
    end
end
for i ← n to 0 ;                                        // (1+n+(n−1))/2
do
    if array[i] > 0 ;                                   // (n−1)/2
    then
        end = i ;                                       // 1
    end
end
for i ← begin to end do
    if currentMax < 0 then
        currentMax := array[i]
    else
        currentMax+ = array[i]
    end
    if currentMax >= maxSum then
        currentSum := currentMax
    end
end
return maxSum;
```

In order to explain how improvement version works, we should analyze number of elementary operations. Every negative number in series from left/right side decrease number of elementary operations which should be performed by 1. Using the improved version of Kadane's algorithm one can assume that additional code (searching interval which contains solution) takes:

$$T(n) = 2 + 2*(\frac{1+n+(n-1)}{2} + \frac{n-1}{2} + 1) = 3n + 3$$

Productivity growth relative to the classical algorithm. From line (2) one can conclude that a loop-cycle costs 4 elementary operations, line (3) shows that a reduction in the range of one real number takes 3 elementary operations. Constants are not taken into account because they are negligible for a large n. From this one can conclude that finding a real number, which is not in the solution interval, allows to reduce the number of operations (by 25%) required to solve the MPS problem.

In the worst case (the first and the last real number in the array are positive) the improved version of Kadane's algorithms will perform 10 more elementary operations (for checking the first and last real number and for two assignments) than the classical algorithm. In the most favourable situation (all real numbers in the array are negative) the improved version of Kadane's algorithm will perform 4/3 elementary operations of the classical algorithm.

Correctness of the results over the classical algorithm. The improvement of the classical algorithm does not change the result which can be achieved by running it. Even in the case of ambiguous solutions, for example in a zero indexed array, such that:

$$C = [-1,\ 2, 1,\ -3, 1, 2,\ -1]$$

where there is one result in the situation of the maximum sum which is 3 and there are 3 possible results in the context of intervals ([1, 2], [1, 5], [4, 5]). The result interval is [1, 5] in both versions of the algorithms.

1.2. Performance tests

Tests were performed on isolated virtual machine. The hardware used for testing was: Intel(R) Core(TM) i5-3320M, 8GB RAM with Debian 64-bit system. There were two environments used to further confirm the results: compiled and interpreted. The following platforms, on which performance was tested, had identical code in terms of semantics:

- C++ (compiler g++ version 4.8.2),
- Ruby (interpreter version 2.1.1p76).

Data for tests were created using standard Ruby "Random" class. Generated integers were saved to text file. One file consists of one data set as follow:

- positive integers: range from 0 to 10,
- negative integers: from -10 to 0,
- mixed integers: from -10 to 10.

The number of integers in sets for C++ and Ruby are different due to how fast the programs written in certain programming languages are. They are as follows:

- C++:
  - 50 000 000 mixed,
  - 12 500 000 negative and 37 500 000 positive,
  - 25 000 000 negative and 25 000 000 positive,
  - 37 500 000 negative and 12 500 000 positive;
- Ruby:
  - 1 000 000 000 mixed,
  - 250 000 000 negative and 750 000 000 positive,
  - 500 000 000 negative and 500 000 000 positive,
  - 750 000 000 negative and 250 000 000 positive.

It is important to notice that in case of negative data we mean negative in series at the begin or/and end of series. For example set of 4 negative and 8 positive can be [-4, -1, -5, -3, 3, 2, 1, 1, 4, 9, 3, 9] or [-2, -9, 3, 1, 6, 7, 4, 3, 2, 9, -1, -3] and can't be [-1, -3, 3, 9, 1, -9, 1, 1, 2, 3, 6, -1].

In this section test results for C++ implementation are presented. Running times of two versions of the algorithm on the same data sets are shown on the graph below (Fig. 1):

- mixed data – decrease in performance by 0,37% ,
- 25% of negative and 75% of positive integers – increase in performance by 8.29%,
- 50% of negative and 50% of positive integers – increase in performance by 23.37%,
- 75% of negative and 25% of positive integers – increase in performance by 28.06%.

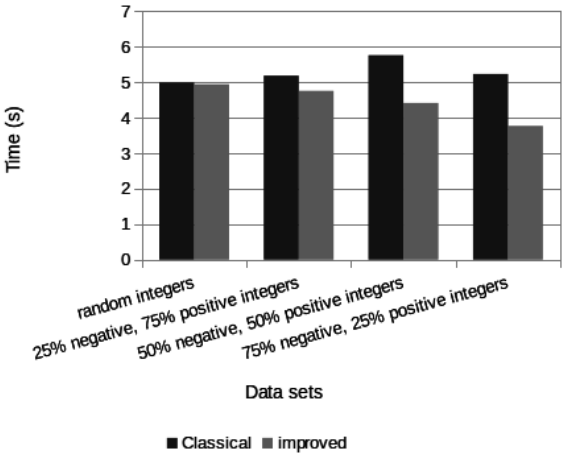

Fig. 1. Classical and improved algorithm run time (C++ implementation) for different data set

Ruby test results were performed in the same manner. They are presented on Fig 2:

- mixed data – decrease in performance by 0,53%,
- 25% of negative and 75% of positive integers – increase in performance by 6.97%,
- 50% of negative and 50% of positive integers – increase in performance by 11.98%,
- 75% of negative and 25% of positive integers – increase in performance by 20.6%.

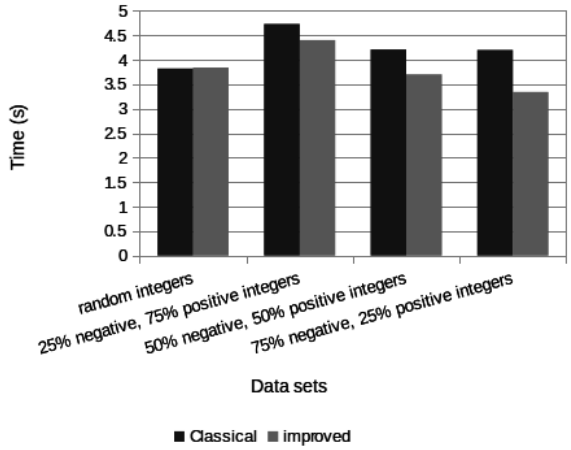

Fig. 2. Classical and improved algorithm run time (Ruby implementation) for different data set

In relation to the results of the performance tests above, the increase can be estimated as shown on Fig. 3. It also converges with the algorithm analysis.

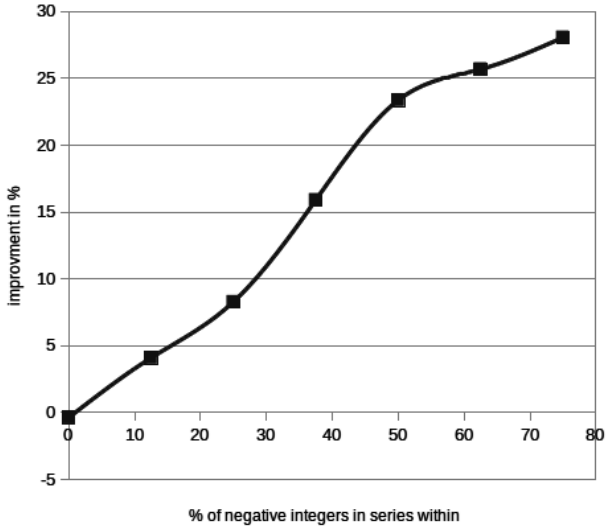

Fig. 3. Estimated improvment

2. Result

In previous section we showed that modified version of Kadane's algorithm increase its performance. Data for test were generated to analyze theoretical improvement. In this section we present that there are some applications of proposed improvement. Among others Kadane's algorithm in two dimensional version is used to find brightest region or region with specific features (dominant color – red, green or blue maximum likelihood estimator of a certain kind of pattern in digital image) is time consume problem. Example of that kind of task is finding brightest (stars) regions on the sky or warmest/coolest place on thermo-graphic image.

To find brightest region on an image, it should be represented as a two dimensional array, where each pixel is luminance values. As a luminance one can assume that this is (for 24-bit images) according to formula [4]:

$$Y = 0.2126R + 0.7152G + 0.0722B$$

Then we should find the area (rectangle) where sum of all pixels is the highest. We can reduce this problem to the maximum subarray problem (MSP) for two dimensional space. For examples if we are given a two-dimensional array a[0..m][0..n], where upper-left corner has coordinates (0,0). The maximum subarray in the following examples is the part of the array with coordinates a[0..1][1..2], which sum is 15.

$$A_{m,n} = \begin{matrix} -3 & 2 & -1 & -4 \\ -8 & 7 & 4 & -1 \\ 0 & -1 & 1 & 5 \\ 0 & -2 & -12 & 7 \end{matrix}$$

The area with maximum sum is row (0,1) and column (1,2).

The best known algorithms which solves MSP is Kadane's algorithm do it in O(n^2) time. In digital images values of pixels are all non negative. The solution of 2D MPS will be the whole array. Before computing the maximum subarray, we should normalize each pixel value by substracting an anchor value. Selection of certain anchor value determinate sensitiveness (size) of region which we want to find.

The whole operation is time consuming, for example for an image 3MP (2,048 pixels × 1536 pixels), which is exactly 3145728 pixels, one can assume that is input data n for our algorithm, time of running is $O(nm^2)$ (qubic time when m = n), the number of operations which algorithm have to do in approximation is $9.89560465 \cdot 10^{12}$.

Today personal computer, which has a two-core processor 2 GHz possesses a theoretical power of about 16GFLOPS, which stems from the formula:

$$FLOPS = cores \cdot clock \cdot \frac{FLOPs}{cycle},$$

Where $\frac{FLOPs}{cycle}$ in equal to 4 in most present microprocessors. If we have such power at our disposal we may solve the above problem in $\frac{9.89560465 \cdot 10^{12}}{16 \cdot 10^{9}} \approx 619\ s$ what is more than 10 minutes. This calculations are approximate, we assume one operation per pixel which is not precise. They do not count real number of operations, but show the scale size of the problem.

2.1. Performance tests for 2D application

We take 3 images from different categories for tests:

- astronomy,
- thermography,
- computed tomography.

All images were 24-bit. In order to find specific (brightest) region on an image, we convert it into gray scale (8 bit) and subtract anchor (about 128) value from each pixel's value. We get two dimensional array with pixels in range (-x to 256 - x, where x is chosen anchor value) as a result of this operation. We can adjust sensitive of searched region by changing parameters: anchor value and luminance formula.

Both versions of Kadane's algorithm were implemeneted in C++ Below we present taken images with selected area which was found and percent of improvement over classical algorithm.

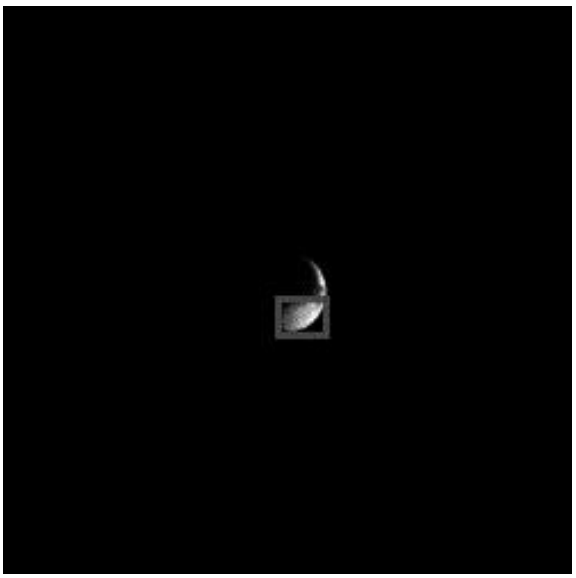

Fig. 4. Astronomy image: 21% faster then classical

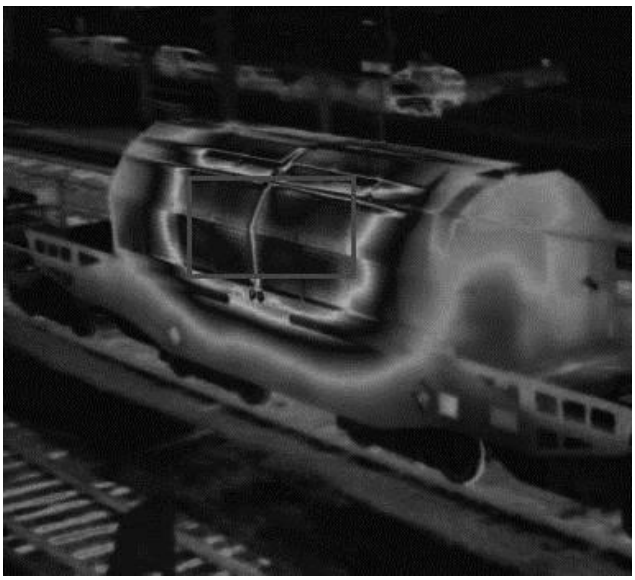

Fig. 5. Thermographic image: 18% faster then classical algorithms

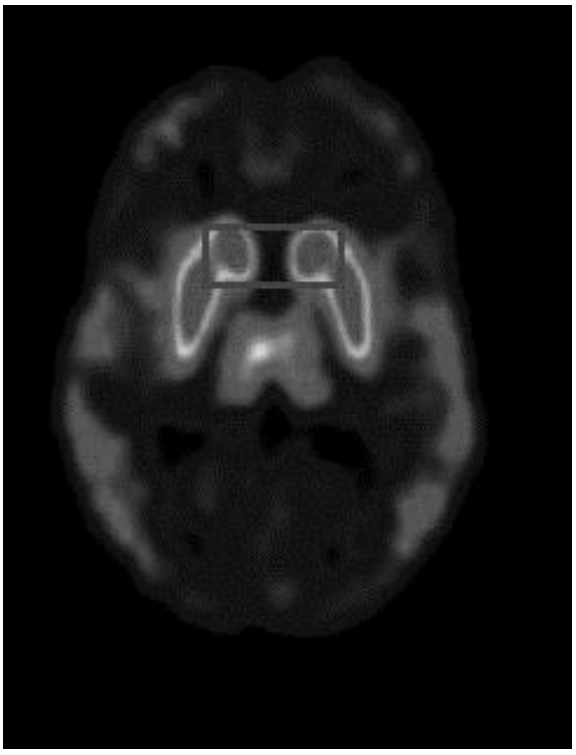

Fig .6. PET image: 5.52% faster than classical algorithms

3. Conclusion

Kadane's algorithm is the state of the art algorithm for solving the maximum subarray problem. One can conclude that the presented improvement can limit the number of elementary operations. The algorithm analysis showed that finding one element of a sequence which does not contain a solution costs 3 units of time and the analysis of the faulty component costs 4 units. The tests prove that with the falling number of elementary operations the performance of the algorithm increases. For the most unfavourable data, the performance is slightly worse in comparison to Kadane's algorithm. In all other situations presented algorithm's runtime is lower. Regardless of the implementation platform (C++, Ruby), a proportional gain in the runtime is achieved. Analogously improvement version of Kadane's algorithm for 2 dimmensional space show that finding the brightest regions can be faster up to 22% then classical Kadane's algorithm.There are applied applications like analysis of DNA sequence or protein sequence, where a significant increase in efficiency is possible. This improvement could be the state of the art for Kadane's algorithm.

Bibliography

[1] Lloyd A.: Longest biased interval and longest non-negative sum interval. Bioinformatics 19, 2003, 1294–1295.
[2] Bae Sung Eun: Sequential and Parallel Algorithms for the Generalized Maximum Subarray Problem. Ph.D. Thesis, University of Canterbury, 2007.
[3] Bentley J.: Programming pearls: algorithm design techniques. Communications of the ACM, 27(9), 1984, 865–873.
[4] BT Series Broadcasting. Parameter values for the HDTV standards for production and international programme exchange BT Series Broadcasting service – volume 5, 2002.
[5] Grenander U.: Pattern analysis. Springer, 1978.
[6] Huang X.: An algorithm for identifying regions of a DNA sequence that satisfy a content requirement. Computer applications in the biosciences: CABIOS 10, 1994, 219–225.
[7] Larson R. C., Odoni A. R.: Urban operations research. Prentice-Hall, New Jersey 1981.
[8] Lin Yaw Ling, Jiang Tao, Chao Kun Mao: Efficient algorithms for locating the length-constrained heaviest segments with applications to biomolecular sequence analysis. Journal of Computer and System Sciences 65, 2003, 570–586.
[9] Perumalla K., Deo N.: Parallel algorithms for maximum subsequence and maximum subarray. Parallel Processing Letters 5(03), 1995, 367–373.
[10] Tokyo IBM: Data Association Mining Rules: Using Algorithms, Optimized and Scheme, Visualization, 1996.
[11] Wang L., Xu Ying: SEGID: Identifying interesting segments in (multiple) sequence alignments. Bioinformatics 19, 297–298, 2003.

M.Sc. Tomasz Rojek
e-mail: trojek@pk.edu.pl

Scientific and didactic assistant at the Institute of Applied Informatics, Faculty of Mechanical Engineering, Cracow University of Technology. Research interests are focused on theoretical computer science, algorithms, computational complexity and image analysis.

otrzymano/received: 15.06.2016 *przyjęto do druku/accepted: 22.11.2017*