# Efficient Algorithms for Subwindow Search in Object Detection and Localization

Senjian An, Patrick Peursum, Wanquan Liu and Svetha Venkatesh
Dept. of Computing, Curtin University of Technology
GPO Box U1987, Perth, WA 6845, Australia.
s.an,p.peursum, w.liu, s.venkatesh@curtin.edu.au

## Abstract

*Recently, a simple yet powerful branch-and-bound method called Efficient Subwindow Search (ESS) was developed to speed up sliding window search in object detection. A major drawback of ESS is that its computational complexity varies widely from $O(n^2)$ to $O(n^4)$ for $n \times n$ matrices. Our experimental experience shows that the ESS's performance is highly related to the optimal confidence levels which indicate the probability of the object's presence. In particular, when the object is not in the image, the optimal subwindow scores low and ESS may take a large amount of iterations to converge to the optimal solution and so perform very slow. Addressing this problem, we present two significantly faster methods based on the linear-time Kadane's Algorithm for 1D maximum subarray search. The first algorithm is a novel, computationally superior branch-and-bound method where the worst case complexity is reduced to $O(n^3)$. Experiments on the PASCAL VOC 2006 data set demonstrate that this method is significantly and consistently faster (approximately 30 times faster on average) than the original ESS. Our second algorithm is an approximate algorithm based on alternating search, whose computational complexity is typically $O(n^2)$. Experiments shows that (on average) it is 30 times faster again than our first algorithm, or 900 times faster than ESS. It is thus well-suited for real time object detection.*

## 1. Introduction

Object detection and localization is one of the most important topics in computer vision, and sliding window methods are widely applied in finding objects [5, 9, 13, 10]. Sliding window methods first train a quality function based on the extracted SIFT[11], SURF [1] or other type of features from the training images, and then apply the quality function on all possible sub-images to find the object by maximizing the quality scores. As $O(n^4)$ subwindows exist for $n \times n$ images, an exhaustive search is computationally prohibitive and thus approximate methods were proposed [6, 7, 18]. To overcome this, recently a branch and bound method, called Efficient Subwindow Search (ESS) [12], was developed to efficiently find the optimal subwindow. When the quality function is linear on the histogram of the extracted features, the quality score of a subwindow just simply sums up the contributions of the extracted features [12]. This allows the subwindow search problem to be reduced to a maximum submatrix problem which finds a region in a matrix with the largest sum of entries, wherein the matrix entry $A[i, j]$ is defined as the sum of the contributions of the features extracted at pixel $(i, j)$. Similarly, for spatial pyramid matching [14], the quality function compares over a few pyramid levels and sums up their contributions [12]. The subwindow search problem can also be formulated as a maximum submatrix search problem but the matrix entry is defined as the sum of the feature contributions for any pyramid level, that is $A[i, j] = \sum_l A_l[i, j]$ where $l$ represents the pyramid levels and $A_l[i, j]$ represents the contributions of the features extracted at pixel $(i, j)$ for comparison in pyramid level $l$.

The maximum subarray search problem is a classical combinational problem which was first discussed by Bentley in his "Programming Pearls" column of *Communications of the ACM* [2, 3] as an example to demonstrate the rules for computational efficiency. The problem is to find a subarray that maximises the sum of the elements within that subarray, as compared to all other possible subarrays. For one-dimensional (1D) arrays, we have an optimal linear time solution known as Kadane's algorithm [2]. By a simple extension, Bentley [3] gave a solution in $O(n^3)$ time for $n \times n$ two dimensional (2D) arrays. Motivated by the linear time solution of the 1D maximum subarray problem, much effort has been made to develop a subcubic (or ideally quadratic) algorithm to solve the 2D maximum subarray problem. Unfortunately, the idea of Kadane's algorithm cannot be extended to 2D arrays since, unlike in 1D arrays, there are multiple ways of ordering the elements in a 2D array. So far, the best known algorithm [16, 17] is of complexity $O(n^3(\log\log n/\log n)^{1/2})$. This sub-cubic algorithm is based on Takaoka's distance matrix multipli-

978-1-4244-3991-1/09/$25.00 ©2009 IEEE

cation algorithm [15]. Whilst it is theoretically important, the algorithm's practical use is limited to very large $n$ (say, larger than a million) since its preprocessing overheads are high. In most practical situations where $n$ is not so large, the subcubic algorithm is actually less efficient than Bentley's cubic algorithm.

Our experiments shows that, although ESS is usually faster than Bentley's cubic algorithm when the confidence score is high (which indicates that the object is in the image with high probability), when the confidence score is low it can be much slower. The upper bound for the number of iterations in ESS is $O(n^4)$ and thus the worst case complexity can be as high as this (though it rarely happens).

In this paper, based on the Kadane's linear-time solution for 1D maximum subarray search, we present two improved algorithms for efficient subwindow search for object detection and localization.

The first one, called Improved ESS (I-ESS), applies the idea of branch-and-bound row-wise, using Kadane's algorithm to compute the bound column-wise. The iterations required are less than $(n(n+1)/2)$, leading to a worst case complexity of $O(n^3)$ (since Kadane's search is linear in $n$). The novelty of this approach lies in the combination of key ideas from Bentley [2, 3] and its application to modify ESS to deliver a significant computational advantage whilst providing the same (optimal) results.

The second technique, called Alternating ESS (A-ESS) is based on alternating search. Given an initial row interval, we optimize the column interval using Kadane's 1D subarray search by summing over the rows in the interval. We then use this column interval to optimize the row interval, summing only over the columns in the found column interval. This alternating row-column optimisation process is repeated until it converges. The iterations required are very small (3 to 6 typically), and thus the computational complexity is quadratic and can be reduced to be linear on the number of feature points for large sparse matrices. However, the approach is not guaranteed to find the globally optimal solution, although empirical results indicate that the errors are not significant.

The layout of the rest in this paper is as follows. In Section 2, we review the formulation of maximum subarray problem and the state-of-art algorithms. Section 3 addresses the novel branch and bound method to solve the 2D subarray problem and Section 4 address the alternating method. The computational complexity is analyzed in Section 5 and experiment results are reported in Section 6 to compare the efficiency of the proposed algorithms to that of ESS and Bentley's algorithm.

## 2. Overview of Maximum Subarray Problem

### 2.1. The Problem Formulation

Given a 1D array $a[1:n]$, the maximum subarray problem is to find a subarray $a[l:r]$ such that the sum of the contained elements is greater than or at least equal to the sum of the elements of any other possible subarrays. Denote

$$\mathrm{SUM}\{a[l:r]\} \triangleq \sum_{i=l}^{r} a[i]. \tag{1}$$

The maximum subarray problem is then to find $l^*$ and $r^*$ so that $\mathrm{SUM}\{a[l^*:r^*]\}$ is equal or greater than $\mathrm{SUM}\{a[l:r]\}$ for any subarray $a[l:r]$ with $[l:r] \subset [1:n]$.

Similarly, for a two-dimensional (2D) array $A[1:n, 1:m]$, which is often called a matrix, the maximum subarray problem is to maximize the function

$$\mathrm{SUM}\{A[t:b, l:r]\} \triangleq \sum_{i=t}^{b}\sum_{j=l}^{r} A[i,j], \tag{2}$$

that is, to find a sub-matrix of $A$ so that its sum is maximal among all $A$'s sub-matrices.

### 2.2. Kadane's Algorithm [2]

If all the elements of $a[1:n]$ are non-positive, it is easy to see that the maximum subarray is the maximal element and the problem can be solved in linear time. Hereafter, we assume that the array has at least one positive element.

**Algorithm 1**: Kadane's Algorithm on Maximum Subarray Search

**Input**: 1D array $a[1:n]$
**Output**: Maximum subarray $a[l:r]$ and its sum $s$
**Initialization**: $(l, r, c, j, s) := (0, 0, 0, 1, 0)$;
**foreach** $i := 1$ *to* $n$ **do**
  $c := c + a[i]$;
  **if** $c > s$ **then**
    $(l, r) := (j, i)$; $s := c$;
  **end**
  **if** $c < 0$ **then**
    $c := 0$; $j := i + 1$ ;
  **end**
**end**

The linear time Kadane's algorithm is based on the following two facts: First, the maximum subarray starts and ends in positive elements; Second, if we start from the first positive element, say $a[l]$, and sum over the subsequent elements until the sum drops negative at $a[r]$, then the optimal subarray is either in $a[l:r]$ and starts from $a[l]$, or in $a[r+1:n]$. Based on these two observations, one can find

a candidate by starting from the first positive element, summing over the subsequent elements and updating the maximum sum until the sum drops negative, say at $a[r]$. By repeating this process from the next positive element after $a[r]$, one can find another candidate– by going through all the elements in the array, one can find the maximum subarray in linear time $O(n)$. The algorithm is summarized in Algorithm 1. This algorithm consists of $n$ additions and at most $2n$ comparisons, so the complexity is around $3n$.

### 2.3. Bentley's Algorithm [3]

Bentley's algorithm simply applies Kadane's algorithm to every possible row interval, summing over the rows in each interval to produce a 1D array for Kadane's algorithm to find the optimal column interval. One of the central ideas of Bentley's algorithm is the prefix sum, which aims to avoid repeating summations when processing subsequent row intervals. Given an array $A[1:m, 1:n]$, the row-wise prefix sum, denoted by $\bar{A}$, is defined as

$$\bar{A}_k \triangleq \sum_{i=1}^{k} A_i, k = 1, 2, \cdots, m \tag{3}$$

where $A_k$ and $\bar{A}_k$ denote the $k$th row of $A$ and $\bar{A}$ respectively. Note that $\bar{A}_{k+1} = \bar{A}_k + A_{k+1}$. The computation of $\bar{A}$ consists of $mn$ additions. A closely related concept is the integral image, which sums over columns and rows rather than just the rows.

With the prefix sum $\bar{A}$, the sum of $A$ over rows in $[t:b]$ can be computed as

$$a = \bar{A}_b - \bar{A}_{t-1} \tag{4}$$

where $a[j] \triangleq \sum_{i=t}^{b} A[i,j], j = 1, 2, \cdots, n$ and $\bar{A}_0$ is defined as a zero vector.

$\big\{[t:b]\big|[t,b] \subseteq [1,m]\big\}$ consists of $m(m+1)/2$ different intervals. Bentley's algorithm simply applies Kadane's algorithm on $a = \bar{A}_b - \bar{A}_{t-1}$ for each interval $[t,b]$ and find the maximum sum. The complexity is of $O(m^2 n)$. If $m > n$, it can be speeded up by simply rotating the matrix so that it has less rows than columns.

### 2.4. Efficient Subwindow Search: A Branch and Bound Method

Here we only describe ESS method for maximum submatrix search which relates the object detection problem when the quality function is linear on the histogram of the extracted features.

One of the central ideas of ESS [12] is to model the full window set in terms of four intervals

$$\mathcal{W}_{full} \triangleq \big\{[t:b, l:r]\big|t \in [1,m], b \in [1,m], l \in [1,n], r \in [1,n]\big\}. \tag{5}$$

The ESS algorithm starts by splitting $\mathcal{W}_{full}$ along its largest interval and computing the bounds of the two disjoint window sets. Then at each iteration, one splits the maximum-bound window set along the largest interval, using a priority queue to efficiently store window sets ordered by their bounds. The procedure is repeated until the maximum-bound window set consists of a single window, which is the optimal solution. After $L$ iterations, $\mathcal{W}_{full}$ is split into $L+1$ disjoint window sets $\mathcal{W}_1, \mathcal{W}_2, \mathcal{W}_{L+1}$. Since $\mathcal{W}_{full}$ has $mn(m+1)(n+1)/4$ members, the ESS algorithm converges and takes at most $(mn(m+1)(n+1)/4 - 1)$ iterations.

For each iteration, the computation of the bound is constant (based on the integral images), and insertions into the priority queue takes $\log(L)$ computations. Hence, the complexity is $O(L\log(L))$, plus the overhead of calculating the integral images (which is $O(n^2)$). Since the upper bound of $L$ is $(mn(m+1)(n+1)/4 - 1)$, the worst case complexity of ESS is $O(n^4 \log n)$ (when $m = n$). The iteration number is closely related to the optimal confidence level. When the optimal confidence level is low and thus the optimal subwindow does not significantly higher than other randomly selected subwindows, it will take a large amount of iterations to converge the optimal solution. In our experiments on PASCAL VOC 2006 data set, we note that, when the optimal confidence score is large (which means that the object is in the image with high probability), the iterations are typically small and sub-linear on the number of pixels, but when the confidence score is small, it may take millions of iterations for some images and ESS performs actually slower than Bentley's algorithm. In the next section, we develop a novel branch-and-bound subwindow search method for object detection and localization.

## 3. A New Branch-and-Bound Method

The main idea of the proposed method is a combination of the key ideas from ESS and Bentley's algorithm. Roughly speaking, we apply the branch-and-bound method to search for the optimal row interval while applying Kadane's algorithm to compute the bound. The optimal column interval is also obtained by Kadane's algorithm when the optimal row interval is found. Next, we will introduce the bound function and its computation, and then explain the splitting procedure and the whole algorithm.

### 3.1. The Bound Function

Let us consider a general interval set

$$\mathcal{I} \triangleq \big\{[t,b]\big|t \in [t_0, t_1], b \in [b_0, b_1]\big\} \tag{6}$$

and its associated window set $\mathcal{W}$ defined as

$$\mathcal{W} \triangleq \big\{[t:b, l:r]\big|[t,b] \in \mathcal{I}, [l,r] \subseteq [1,n]\big\} \tag{7}$$

For convenience, we will call $[t_0, t_1]$ the t-interval and $[b_0, b_1]$ the b-interval. We also assume that

$$t_0 \le t_1, t_0 \le b_0, t_1 \le b_1. \tag{8}$$

without loss of generality. For a non-empty interval set $\mathcal{I}$ defined in (6), if the condition (8) does not hold, one can always find $(\bar{t}_0, \bar{t}_1, \bar{b}_0, \bar{b}_1)$ satisfying (8) so that $\mathcal{I}$ can be described as $\mathcal{I} = \big\{[t,b]\big|t \in [\bar{t}_0, \bar{t}_1], b \in [\bar{b}_0, \bar{b}_1]\big\}$.

In order to define the upper bound for

$$\max_{[t:b,l:r]\in\mathcal{W}} \mathrm{SUM}\big\{A[t:b,l:r]\big\} \tag{9}$$

we need to introduce two complementary arrays $a^+[1:n]$ and $a^-[1:n]$ which are defined as

$$\begin{array}{l} a^+(j) = \sum_{k=t_0}^{b_1} A^+[k,j], j = 1,2,\cdots,n \\ a^-(j) = \left\{ \begin{array}{ll} 0, j = 1,2,\cdots,n, & \text{if } b_0 < t_1; \\ \sum_{k=t_1}^{b_0} A^-[k,j], j = 1,2,\cdots,n, & \text{otherwise.} \end{array} \right. \end{array} \tag{10}$$

where $A^+$ and $A^-$ are defined as

$$A^+[i,j] = \max\big(0, A[i,j]\big), A^-[i,j] = \min\big(0, A[i,j]\big). \tag{11}$$

Note that $A^+$ consists of only the $A$'s positive elements while $A^-$ consists of only the $A$'s negative elements (in addition to zero elements).

Then, for any $[t,b] \in \mathcal{I}$, we have $[t,b] \subseteq [t_0, b_1]$ and $[t,b] \supseteq [t_1, b_0]$ and therefore

$$\begin{array}{rcl} a^+(j) & \ge & \sum_{k=t}^{b} A^+[k,j] \\ a^-(j) & \ge & \sum_{k=t}^{b} A^-[k,j] \end{array} \tag{12}$$

which implies $a^+[j] + a^-[j] \ge \sum_{k=t}^{b} A[k,j]$. Note that this is true for any $[t,b] \in \mathcal{I}$ and $j = 1,2,\cdots,n$, we have

$$\begin{array}{l} \max\limits_{[t:b,l:r]\in\mathcal{W}} \mathrm{SUM}\big\{A[t:b,l:r]\big\} \\ \le \max\limits_{[l,r]\in[1,n]} \mathrm{SUM}\big\{a^+[l:r] + a^-[l:r]\big\}. \end{array} \tag{13}$$

We define

$$g(\mathcal{I}) \triangleq \max_{[l,r]\in[1,n]} \mathrm{SUM}\big\{a^+[l:r] + a^-[l:r]\big\} \tag{14}$$

as a bound function since $g(\mathcal{I})$ is an upper bound for $\mathrm{SUM}\big\{A[t:b,l:r]\big\}$ with windows within $\mathcal{W}$ and $\mathcal{W}$ is directly related to $\mathcal{I}$ (see (7)).

The right side optimization in (14) is actually a 1D maximum subarray search problem and thus the bound $g(\mathcal{I})$ can be found in linear time by applying Kadane's algorithm.

Now, we show that if $\mathcal{I}$ consists of only one interval, i.e., $t_0 = t_1, b_0 = b_1$, the bound $g(\mathcal{I})$ is achieved by $\mathrm{SUM}\big\{A[t_0 : b_0, l_* : r_*]\big\}$ where $[l_*, r_*]$ is the solution of the right side maximization problem in (14). Since $\mathcal{I}$ contains only one member, the largest and the smallest size intervals are identical and thus $a^+ + a^- = a$ where $a[j] = \sum_{i=t_0}^{b_0} A[i,j], j = 1,2,\cdots,n$. Therefore the equality holds in (13) and this implies that $g(\mathcal{I})$ is achieved by $\mathrm{SUM}\big\{A[t_0 : b_0, l_* : r_*]\big\}$ and the claim is true.

Hence, if we can split the full interval set

$$\mathcal{I}_{full} \triangleq \big\{[t,b]\big|t \in [1,m], b \in [1,m]\big\} \tag{15}$$

into a series of disjoint interval sets $\mathcal{I}_{full} = \cup_{k=1}^{L} \mathcal{I}_k$ and the maximum-bound interval set $\mathcal{I}_L$ consists of only one interval, then we have 1). $g(\mathcal{I}_L)$ is achieved by $\mathrm{SUM}\big\{A[t_0 : b_0, l_* : r_*]\big\}$ where $[t_0, b_0]$ is the unique interval in $\mathcal{I}_L$ and $[l_*, r_*]$ is the solution of Kadane's search in computing the bound $g(\mathcal{I}_L)$; 2). $g(\mathcal{I}_L) \ge g(\mathcal{I}_k)$ for any $k < L$. Based on these two statements and note that $g(\mathcal{I}_k)$ is an upper bound for $\mathrm{SUM}\big\{A[t:b,l:r]\big\}$ with windows within

$$\mathcal{W}_k \triangleq \big\{[t:b,l:r]\big|[t,b] \in \mathcal{I}_k, [l,r] \subseteq [1,n]\big\} \tag{16}$$

and the full window set

$$\mathcal{W}_{full} \triangleq \big\{[t:b,l:r]\big|[t,b] \subseteq [1,m], [l,r] \subseteq [1,n]\big\} \tag{17}$$

equals $\cup_{k=1}^{L} \mathcal{W}_k$, we conclude that $A[t_0 : b_0, l_* : r_*]$ is the maximum subarray. i.e., the unique interval in the maximum-bound interval set is the optimal row interval and the optimal column interval is the solution of Kadane's algorithm in computing the maximum bound $g(\mathcal{I}_L)$.

In order to compute the sequences $a^+$ and $a^-$ efficiently for each iteration, we computes the row-wise prefix sum $\bar{A}^+$ and $\bar{A}^-$ in advance which are defined as

$$\begin{array}{rcl} \bar{A}_i^+ & = & \sum_{k=1}^{i} A_k^+ \\ \hat{A}_k^- & = & \sum_{k=1}^{i} A_k^- \end{array} \tag{18}$$

where $\bar{A}_i^+, \bar{A}_i^-, A_i^+$ and $A_i^-$ denote the $i$th row of the matrices $\bar{A}^+, \bar{A}^-, A^+$ and $A^-$ respectively.

With these two row-wise prefix sum matrices,

$$\begin{array}{rcl} a^+ & = & \bar{A}_{b_1}^+ - \bar{A}_{t_0-1}^+, \\ a^- & = & \left\{ \begin{array}{ll} \mathbf{0}, & \text{if } b_0 < t_1; \\ \bar{A}_{b_0}^- - \bar{A}_{t_1-1}^-, & \text{otherwise.} \end{array} \right. \end{array} \tag{19}$$

and thus can be computed in $O(n)$ time in any iteration.

### 3.2. The Splitting

For an interval set $\mathcal{I}$, we split it into two disjoint nonempty sets along the t-interval or b-interval (the one with larger size), say $[t_0, t_1] = [t_0, \tilde{t}] \cup [\tilde{t}, t_1]$ where $\tilde{t}$ is the largest integer not greater than $(t_0 + t_1)/2$. Then $\mathcal{I}$ is split into two disjoint nonempty sets $\mathcal{I} = \mathcal{I}_1 \cup \mathcal{I}_2$ where

$$\begin{array}{rcl} \mathcal{I}_1 & \triangleq & \Big\{[t,b]\big|t \in [t_0, \tilde{t}], b \in [b_0, b_1]\Big\} \\ \mathcal{I}_2 & \triangleq & \Big\{[t,b]\big|t \in [\tilde{t}+1, t_1], b \in [\max(\tilde{t}+1, b_0), b_1]\Big\}. \end{array} \tag{20}$$

Accordingly, $\mathcal{W}$ is split into $\mathcal{W}_1 \cup \mathcal{W}_2$ where

$$\begin{aligned} \mathcal{W}_1 &\triangleq \Big\{[t:b, l:r] \Big| [t,b] \in \mathcal{I}_1, [l,r] \subseteq [1,n]\Big\}, \\ \mathcal{W}_2 &\triangleq \Big\{[t:b, l:r] \Big| [t,b] \in \mathcal{I}_2, [l,r] \subseteq [1,n]\Big\}. \end{aligned} \tag{21}$$

### 3.3. The Algorithm

The algorithm starts by splitting the full interval set $\mathcal{I}_{full}$ and computing the bounds of the two disjoint interval sets. Then at each iteration, one splits the maximum-bound interval set along the t-interval or b-interval (the one with larger size), using a priority queue to efficiently store the interval sets ordered by their bounds. The procedure is repeated until the maximum-bound interval set consists of a single interval, which is actually the optimal row interval. After $L$ iterations, $\mathcal{I}_{full}$ is split into $L+1$ disjoint interval sets $\mathcal{I}_1, \mathcal{I}_2, \mathcal{I}_{L+1}$. Since $\mathcal{I}_{full}$ has $m(m+1)/2$ members, the proposed algorithm converges, taking at most $(m(m+1)/2 - 1)$ iterations. The algorithm is summarized as follows. The input and output are the same as that in Bentley's algorithm and thus omitted.

**Algorithm 2**: Improved Efficient Subwindow Search (I-ESS)

**Input**: Matrix $A[1:m, 1:n]$
**Output**: Maximum submatrix $A[t:b, l:r]$
**Ini.**: Set $\mathcal{I}_{full}$ as the max-bound interval set, set the priority queue $P$ empty;
**Preproc.**: Compute the prefix sum matrices $\bar{A}^+, \bar{A}^-$;
**while** *Size of max-bound interval set is not 1* **do**

1. Split max-bound interval set into two disjoint sets along the t-interval or b-interval;
2. Compute bounds of two new interval sets using Kadane's algorithm;
3. Push both interval sets onto $P$, ordered by bounds;
4. Pop maximum-bound interval set from $P$;

**end**

For each iteration, the computation of the bound is $O(n)$ and insertions into the priority queue take $O(\log(L))$ ($<< m < n$). Hence, the overall complexity is $O(Ln)$ (disregarding the small $\log L$ term). Since $L \leq m(m+1)/2$, I-ESS is more efficient than Bentley's algorithm if $L$ is not very close to $m(m+1)/2$, which is typically the case. And similarly to ESS, the larger the optimal confidence score, the smaller $L$ will be. However, iteration number of I-ESS is less dependent on the optimal confidence level than that of ESS since I-ESS only applies branch and bound methods along rows.

## 4. Alternating Search: An Approximate Faster Algorithm

Suppose $[t^* : b^*, l^* : r^*]$ is the optimal sub-window which maximizes the function $\mathrm{SUM}\big\{A[t:b, l:r]\big\}$. Then $[t^* : b^*, l^* : r^*]$ satisfies the following two conditions:

1. $[t^* : b^*]$ is the solution of the maximum subarray problem for 1D array $a[1:m]$ where

$$a[i] \triangleq \sum_{j=l^*}^{r^*} A[i,j]. \tag{22}$$

2. $[l^* : r^*]$ is the solution of the maximum subarray problem for 1D array $b[1:n]$ where

$$b[j] \triangleq \sum_{i=t^*}^{b^*} A[i,j]. \tag{23}$$

However, the subwindows satisfying the above two conditions may not be the globally optimal solution. Instead, locally optimal solutions may be found.

Based on the above observations and the fact that there is a linear time algorithm to solve the 1D maximum subarray problem, we propose the following alternating search algorithm A-ESS. We start by initializing $[t:b] = [1:m]$ and optimize the column interval $[l:r]$ by summing over the rows and applying Kadane's algorithm. Then we fix the column interval and optimize the row interval $[t:b]$ by summing over the columns (within the column interval) and applying Kadane's algorithm to this. This alternating optimization of the row and column intervals is repeated until convergence. The convergence is guaranteed since the maximum sum function is bounded and non-decreasing for each iteration. More precisely, let $[t_k : b_k]$ be the optimal row interval obtained in $k$th iteration with fixed column interval $[l_{k-1} : r_{k-1}]$ (which is obtained from the last iteration). Let

$$f_k \triangleq \mathrm{SUM}\{A[t_k : b_k, l_{k-1} : r_{k-1}]\}. \tag{24}$$

Note that the summation operation of a matrix is defined in section 2.1. In the next iteration, the row interval $[t_k : b_k]$ is fixed and we optimize the column interval. Let $[l_{k+1} : r_{k+1}]$ be the optimal solution. Then we have

$$\begin{aligned} f_{k+1} &\triangleq \mathrm{SUM}\{A[t_k : b_k, l_{k+1} : r_{k+1}]\} \\ &= \max_{l\in[1,n], r\in[1,n]} \mathrm{SUM}\{A[t_k : b_k, l : r] \\ &\geqslant \mathrm{SUM}\{A[t_k : b_k, l_{k-1} : r_{k-1}]\} \\ &= f_k \end{aligned} \tag{25}$$

and therefore the score function is non-decreasing for each iteration. Also, the optimal score is bounded by the sum of all the matrix's positive elements, and hence the alternating search algorithm is convergent. However, it may converges

to a locally optimal submatrix instead of the global optimal solution. The algorithm is summarized as follows. The input and output are the same as that in I-ESS algorithm and thus omitted.

**Algorithm 3**: Alternating ESS (A-ESS)

Initialization: Set $(t, b) = (1, m)$ and $g = 1$;
**while** $g > 0$ **do**

1. Compute $a[j] = \sum_{i=t}^{b} A[i, j]$, $j = 1, 2, \cdots, m$;
2. Apply Kadane's algorithm on $a$ and find the optimal column interval $[l, r]$ and the maximum $s_1$;
3. Compute $b[i] = \sum_{j=l}^{r} A[i, j]$, $i = 1, 2, \cdots, n$;
4. Apply Kadane's algorithm on $b$ and find the optimal row interval $[t, b]$ and the maximum $s$;
5. Set the gain $g = s - s_1$.

**end**

Similarly, one can also start the algorithm by initializing the column interval $[l, r] = [1, n]$ rather than starting from the rows. In our implementation, we apply both initializations and choose the solution that yields the larger sum. This can help reduce the problem of encountering a local maxima.

In step 2 and step 4, Kadane's algorithms takes $O(m+n)$ operations. For the computations in step 1 and step 3, we recommend to compute, in advance, the row-wise and column-wise prefix sum matrices $\bar{A}$ and $\hat{A}$ where $\bar{A}[i, 1 : n] = \sum_{k=1}^{i} A_k[k, 1 : n]$ and $\hat{A}_i[1 : m, i] = \sum_{k=1}^{i} A[1 : m, k]$. Then the vectors $a$ in step 1 and $b$ in step 3 can be computed as $a = \bar{A}[b, :] - \bar{A}[t - 1, :]$ and $b = \hat{A}[:, r] - \hat{A}[:, l - 1]$ which involve $O(n + m)$ operations. Hence A-ESS involves $O(mn)$ computations in computing the prefix sum matrices and $O(l(n + m))$ computations in searching where $l$ is the number of iterations. Since there are at most $(m + 1)m/2$ row intervals and at most $(n + 1)n/2$ column intervals, the iteration number is less than any of them. Hence the computation complexity may vary from $O(n^2)$ to $O(n^3)$. However, A-ESS typically takes 3 to 6 iterations for convergence in our experiments for a large variation of matrix sizes and thus the complexity of A-ESS is dominated by the computations of the prefix sum matrices and thus quadratic. We conjecture that the iteration number depends on the statistical property rather than the size of the matrix.

In the case that the matrix is sparse and the matrix size is so large that the quadratic algorithm is expensive, one may choose to compute $a$ and $b$ directly from the matrix's nonzero elements instead of using the prefix sum matrices. Since the iteration number is typically small and independent of the matrix size, the time and memory complexity will be reduced to be linear on the number of nonzero entries of the sparse matrix.

## 5. Complexity Analysis

Except for Bentley's algorithm, the computational complexity of the other three algorithms depends on their iterations and so is difficult to estimate. However, we can estimate the complexity for the best and worst cases. For an $n \times n$ matrix, the time complexity and memory requirement are reported in Table 1.

Table 1. Time Complexity and Memory Requirement for $n \times n$ Matrices.

| | Time Complexity | | Memory Requirement | |
|---|---|---|---|---|
| Methods | Best Case | Worst Case | Best Case | Worst Case |
| BENT | $O(n^3)$ | $O(n^3)$ | $O(n^2)$ | $O(n^2)$ |
| ESS | $O(n^2)$ | $O(n^4)$ | $O(n^2)$ | $O(n^4)$ |
| I-ESS | $O(n^2)$ | $O(n^3)$ | $O(n^2)$ | $O(n^2)$ |
| A-ESS | $O(n^2)$ | $O(n^3)$ | $O(n^2)$ | $O(n^2)$ |

We summarize the major points about the complexity analysis as follows.

1. Bentley's algorithm has the same complexity for best and worst cases. The complexity depends only on the matrix size.
2. ESS varies the most for best and worst cases. Both the time and memory complexity vary from $O(n^2)$ to $O(n^4)$. The performance variation of ESS can be very large for different matrices.
3. Both memory and time complexity of I-ESS are less than or equal to that of the Bentley's algorithm and thus we expect that I-ESS performs better than the Bentley's algorithm for most matrices.
4. Though the best and worst complexity of A-ESS is the same as that of I-ESS, its iteration number is usually much smaller and independent of the matrix sizes. Moreover, for large sparse matrices, the computation complexity can be reduced to be linear on the number of matrix's nonzero elements.

## 6. Experimental Results

The experiments were conducted on the PASCAL VOC 2006 database [8] and based on the classifiers generated by the structured output regression method [4]. So as to ensure a fair and unbiased comparison to ESS we obtained and used the actual feature point data (including weights) and their ESS code[1] from the authors of [4, 12].

[1] Kindly supplied by the authors of [4, 12] at http://christoph.lampert.googlepages.com/software

All experiments [2] were conducted on a standard desktop PC (Intel Core 2 Duo 3.0GHz E8400) running Windows XP, compiled using Visual Studio .NET 2005. The PASCAL VOC 2006 database consists of 5304 images containing 9507 objects from 10 categories. For each image and each category, we have a 2D array. In total we conduct 53040 subwindow searches for each method and compare their efficiency.

### 6.1. Search Time Comparison

The performance of Bentley's algorithm (BENT), ESS and the two proposed methods I-ESS, A-ESS, are shown in Table 2 and Figure 1. Table 2 shows the clear computational advantage of the two proposed algorithms. On average, I-ESS is about 30 times and 28 times faster than ESS and Bentley's algorithm respectively, and the A-ESS is a further 30 times faster than I-ESS.

The histograms of ratios of the search times of ESS, I-ESS and A-ESS against Bentley's algorithm for all 53,040 subwindow searches are presented in Figure 1, which demonstrates that I-ESS and A-ESS are consistently faster than Bentley's algorithm. Note that I-ESS and A-ESS are generally faster and have lower performance variance than ESS. While ESS can be 70 times faster than Bentley's algorithm in some cases, it can also be more than 10 times slower than Bentley's algorithm in other cases. This coincides with our analysis on ESS that its upper bound of iterations is $O(n^4)$ and so the variation of its performance can be very large. Roughly speaking, ESS is usually fast when the object is clearly in the image yielding high confidence scores in sub-window search. However, if the image does not contain the object, the optimal confidence score may be low and ESS may be significantly slower. Figure 2 shows the search times of ESS, I-ESS and A-ESS against the optimal confidence levels. The confidence levels are quantized into 15 grids and the search time are averaged in each grid. From Figure 2, one can see that the search time decreases as optimal confidence level increases for all the three algorithms. However, ESS is much more dependent on the confidence levels than I-ESS and A-ESS.

Table 2. Search Time Comparison on PASCAL VOC 2006.

| | CPU Time (milliseconds) | | |
|---|---|---|---|
| Methods | Average | minimum | Maximum |
| BENT | 489 | 48 | 1342 |
| ESS | 538 | 4.13 | 25440 |
| I-ESS | 17.6 | 1.1 | 129 |
| A-ESS | 0.58 | 0.15 | 1.92 |

[2] Codes available at http://impca.cs.curtin.edu.au/downloads/software.php

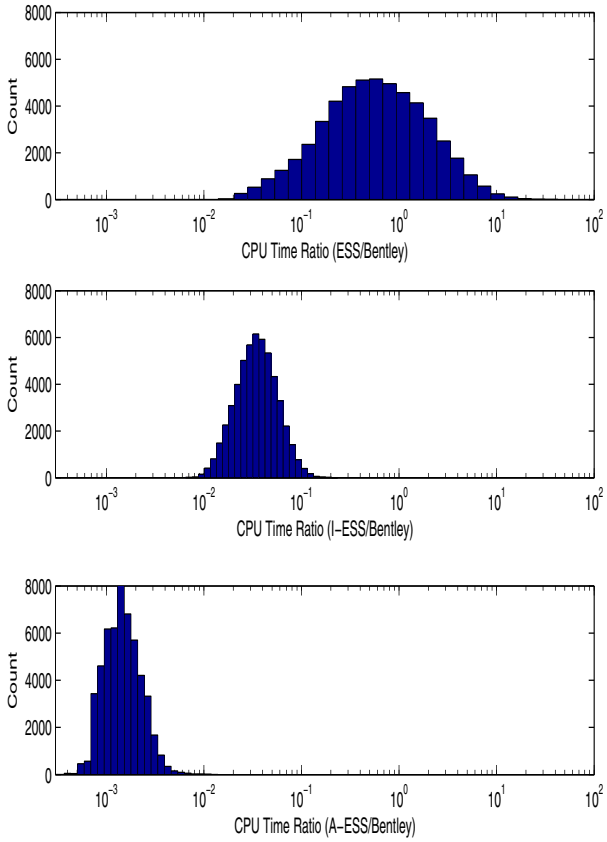


Figure 1. Search Time Comparison with Bentley as a Base Algorithm.

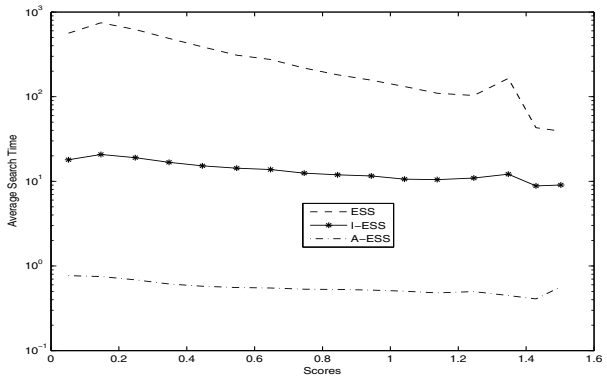


Figure 2. Search Time Versus Optimal Confidence Levels.

### 6.2. A-ESS's Accuracy

In all 53040 subwindow searches by A-ESS, there are around 2.7% of the cases with a confidence score that differs by more than 5% from ESS. The scores of A-ESS of these 2.7% worst cases are reported in Figure 3 with a comparison to the optimal scores. From Figure 3, one can see that, the significant errors are encountered only when the

optimal score is low ( $\leq 0.2$) and thus does not effect the detection performance. This is because the optimal score is already low enough to show the non-presence of the object and it does not matter if A-ESS finds a subwindow with a lower score. Note that Figure 3 only reports the worst $2.7\%$ cases and the maximum score $0.35$ is also low enough to show the non-presence of the object. The highest score for all the cases is 1.7025.

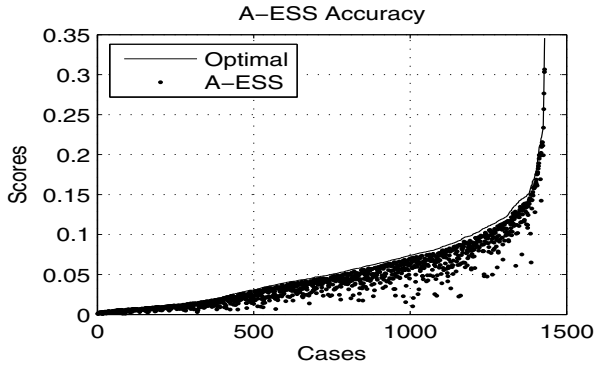


Figure 3. A-ESS's Error Analysis. This figure reports the cases with more than $5\%$ relative errors and the cases are sorted by increasing optimal scores. Note that the maximum score $0.35$ in these cases is low enough to show the non-presence of the object. The highest score for all the cases is 1.7025

## 7. Conclusion

We have proposed a novel branch and bound method which is consistently and significantly faster than the state-of-art algorithms for maximum subwindow search. An even faster, sub-quadratic time method called alternating efficient subwindow search (A-ESS) is also proposed, although it may find only a local maxima. With features available, A-ESS always completes object detection within two milliseconds for the images in the PASCAL VOC 2006 database. Thus it is highly suitable for use in online applications.

The proposed algorithms are applicable in applications of objection detection when quality function is linear on the extracted features. They can also be applied for linear spatial pyramid machining. However, the proposed algorithms are not applicable for nonlinear quality functions so far. We are now working to extend the ideas to speed up subwindow search when the quality function is some typical nonlinear kernel functions.

## Acknowledgements

The authors would like to thank Dr Christoph H. Lampert for providing us the extracted features and cluster IDs of the images in the PASCAL VOC 2006 database, and the classifier weights based on the structured learning.

## References

[1] H. Bay, T. Tuytelaars, and L. J. Gool. SURF: Speeded Up Robust Features. In *Proceedings of ECCV*, 2006. 1

[2] J. Bentley. Programming pearls: algorithm design techniques. *Commun. ACM*, 27(9):865–873, 1984. 1, 2

[3] J. Bentley. Programming pearls: perspective on performance. *Commun. ACM*, 27(11):1087–1092, 1984. 1, 2, 3

[4] M. B. Blaschko and C. H. Lampert. Learning to localize objects with structured output regression. In *Proceedings of ECCV*, 2008. 6

[5] A. Bosch, A. Zisserman, and X. Munoz. Representing shape with a spatial pyramid kernel. In *Proceedings of the 6th ACM international Conference on image and video retrieval*, pages 401–408, 2007. 1

[6] O. Chum and A. Zisserman. An exemplar model for learning object classes. In *Proceedings of CVPR*, 2007. 1

[7] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Proceedings of CVPR*, 2005. 1

[8] M. Everingham, A. Zisserman, C. K. I. Williams, and L. Van Gool. The PASCAL Visual Object Classes Challenge 2006 (VOC2006) Results. http://www.pascal-network.org/challenges/VOC/voc2006/results.pdf. 6

[9] V. Ferrari, L. Fevrier, F. Jurie, and C. Schmid. Groups of adjacent contour segments for object detection. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 30(1):36–51, 2008. 1

[10] M. Fritz and B. Schiele. Decomposition, discovery and detection of visual caregories using topic models. In *Proceedings of CVPR*, 2008. 1

[11] L. D. G. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, 2004. 1

[12] C. H. Lampert and M. B. Blaschko. Beyond sliding windows: Object localization by efficient subwindow search. In *Proceedings of CVPR*, 2008. 1, 3, 6

[13] H. Rowley, S. Baluja, and T. Kanade. Human face detection in visual scene. In *Advances in Neural Information Processing Systems (NIPS96)*, pages 875–881, 1996. 1

[14] L. S., C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Proceedings of CVPR*, 2006. 1

[15] T. Takaoka. A new upper bound on the complexity of the all pairs shortest paths probelm. *Inform. Process. Lett.*, 43(4):195–199, 1992. 2

[16] T. Takaoka. Efficient algorithms for the maximum subarray problem by distance matrix multiplication. In *Proc. CATS 2002, ENTCS*, volume 61, pages 191–200, 2002. 1

[17] H. Tamaki and T. Tokuyama. Algorithms for the maximum subarray problem based on matrix multiplication. In *Proc. SODA 1998*, 1998. 1

[18] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings CVPR*, 2001. 1