A Project Report

On

Computational attention modeling on webpages

BY

B Rishi Saimshu Reddy

2018A7PS0181H

Under the supervision of

Prof. Sandeep Vidyapu

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS OF

CS F366: LABORATORY PROJECT

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)

HYDERABAD CAMPUS

(November 2020)

Acknowledgment

I would like to express my gratitude towards Prof. Sandeep Vidyapu for providing me with the opportunity to work on this project and learn and explore new areas in Human-Computer Interaction. I would like to thank him for providing me with proper guidance, his valuable insights, and help whenever I needed them.

Lastly, I acknowledge BITS Pilani and the AUGSD for their efforts in providing us with these excellent opportunities to work on various projects under professors in our field of interest to learn, grow and expand our skills.

Birla Institute of Technology and Science-Pilani,

Hyderabad Campus

Certificate

This is to certify that the project report entitled "**Computational attention modeling on webpages**" submitted by Mr. Bandi Rishi Saimshu Reddy (ID No. 2018A7PS0181H) in partial fulfillment of the requirements of the course CS F366, Laboratory Project Course, embodies the work done by him under my supervision and guidance.

Date: **(Prof. Sandeep Vidyapu)**

BITS- Pilani, Hyderabad Campus

Abstract

Screen readers of today read a webpage in the order as written in the source code. Due to this reason, the process of accessing the web becomes cumbersome to for example, partially/completely blind people (as it may go through many unnecessary web page elements before coming to the required part) who depend on the Screen reader for the information on the webpage. For similar reasons, eye-tracking studies and research on clustering users' scanpaths on web pages to improve internet accessibility has become very important. It allows a developer to design and develop better web pages and improve users' experience in accessing the web under constrained environments, for visually disabled users, etc.

The Scanpath Trend Analysis (STA) algorithm is used to get a trending scanpath on a webpage by clustering the recorded scanpaths of multiple users. Though STA algorithm tries to solve the many issues that other contemporary methods have, but it isn't perfect itself. This report is a work on the summary on STA; Its working, application, strengths and weaknesses and future works based on the STA algorithm.

Contents

Introduction

While physical stores may need to make "reasonable renovations" for example, introducing ramps at entrances and exits or including signage written in Braille to make their place of business more accessible for disabled individuals, accomplishing providing accessibility on the internet depends on special technical solutions. Innovations in assistive technologies like screen readers or magnifiers can enable visually impaired individuals to use the internet. However, these advanced assistive technologies work best with websites planned with their usefulness in mind; in any other case, information gets more difficult to comprehend for the end-user.

Coming to Screen readers, they have been in existence since the past 30 years. A screen reader is a program that examines the design and content of a site and gives a text to speech translation. The user can set the playback speed and other commands allow users to skip from heading to heading, visit links, and do other essential tasks. Much like how a person with vision visually scans a website to find the section they want to read, a blind person can do the same with their screen reader—as long as the website's content has been coded with proper header tags. The problem that arises with these Screen readers is that they read a screen in the order as written in the web page's source code. For a blind user using a screen reader for assistance has to often listen to a lot of unnecessary information before the user actually reaches the part of the page where the required information is at. This becomes a cumbersome process when there are multiple pages the user wishes to browse through for a small task which a user with sight might complete within a short time.

This lead to many eye-tracking studies trying to create technologies that would cluster the scanpath of multiple users to create a so called "trending scanpath"

representing a general scanpath that's very close to all the collected individual scanpaths. There have been many technologies that were brought forward as a solution. Unfortunately, the resulting scanpath from these algorithms and technologies was reductionist in nature (losing a Area of Interest even though most users had that in their scanpath) and were of no use in processing. They were not usable to co-relate the resulting Areas of Interest in the scanpath with the underlying code to enable a screen reader to follow that order. They also overlooked the duration spent at each Area of Interest as a valid parameter in determining what visual elements are part of the trending scanpath.

Scanpath Trend Analysis (STA)algorithm

Compared to the other solutions for traditional screen readers' issues as discussed above, Eraslan et.al(2016) proposed the STA algorithm, which has a different approach to the problem at hand. The STA algorithm was developed to discover trending scanpaths based on scanpaths of multiple web users in terms of visual elements of web pages. It clusters the scanpaths by arranging these visual elements based on their overall positions in the individual scanpaths on which the trending scanpath is constructed. Thus, in this way, STA algorithm not only considers the elements which were visited by majority of the users, but it is also considered when visited in any order. \

The stages of STA algorithm

Preliminary stage:

Input: Extended visual elements, A series of fixations for each user.

(P.T.O)

for each *user* **do**
for each *fixation* **do**
for each *extended visual element* **do**
Find the elements where the fixations are located
end
if *the fixation is located in more than one extended visual element* **then**
The closest element is added to the individual scanpath with the fixation duration.
else if the fixation is located in one extended visual element **then**
The element is added to the individual scanpath with the fixation duration
end
end
end

return *Individual Scanpaths represented in terms of the visual elements and fixation durations*

Output: Individual Scanpaths represented in terms of the visual elements and fixation durations

In the preliminary stages, we generate the indivual scanpaths. Generation of a Trending scanpath starts with segmenting the webpage into visual elements first. The segmentation of webpages is done using the extended and the improved version of the Vision Based Page Segmentation algorithm (ViPS). The VIPS algorithm is used here because it segments the web page into elements and correlates it to the underlying code. Following segmentation of the webpage, individual scanpaths wrt to the visual elements obtained above is generated by relating the fixations of a user obtained using eye tracking data to the visual elements. We store a scanpath in terms of the visual elements visited in order and their specific durations. This stage has a time complexity of $O(n^6)$ attributed to resolving the fact about where the fixations lie in case of it overlapping with more than one visual element. This stage is the slowest step in the whole process of generating the trending scanpath.

First pass:

Input: An instance, Individual Scanpaths in Terms of Visual Element Instances

if *all individual scanpaths share the instance OR it gets at least the same attention as the shared instances* **then**
The instance is marked as a trending visual element instance
else
The instance is marked to be removed from the individual scanpaths
end

return *The marked instance*

Output: The marked Instance

After the **individual scanpaths have been obtained through the preliminary stage, the next step in the STA algorithm is finding the trending visual elements using the occurrence frequency of the scanned visual elements.**

Taking the individual scanpaths, dependening on the length(duration) of each fixation we number the different visual elements in the scan path to represent the impacts of the length of an fixation on the trending scanpath and to keep the numbering consistent over all scanpaths. We have tended to the issue of not considering the length of fixation by doing the above. From here, we move ahead with identifying the trending elements by following certain criteria because of which the issue of reductionist output is eliminated. **In the STA algorithm, an element is considered as a trending element when it shared by all the individual scapaths. The algorithm also analyses the occurrence frequencies and the durations of a visual elements and its fixation in individual scanpaths to determine a threshold value for the same.** Thus, when a visual element instance is not shared by all the individual scanpaths but at least gets the same amount of attention when compared to other shared visual elements, it is considered as an trending element by the STA algorithm. This stage has a time complexity of $O(n^3 \log n)$.

Second pass:

Using the trending visual elements found in the first pass a trending scanpath is constructed by combining individual scanpaths. These visual elements are located in the trending scanpath depending on their overall location in the individual scanpaths.

The appropriate location/position of the trending visual elements are determined by their total priority value. In an individual scanpath, the total priority value of an element is the sum total of that element's priority values in all the individual scanpaths. These elements are then placed into their positions in the trending scanpath depending on their total priority value, where the highest total priority valued visual element is positioned at the starting. The STA algorithm also uses their frequency of occurrences and the duration of fixation over it to position them in the scanpath as the secondary criteria. The second pass of the STA algorithm has a time complexity of $O(n^3)$ and at the end of this stage, a trending scanpath generated by clustering multiple users' scanpath data is returned.

STA Implementation

> After importing required libraries, the following parameters are also passed for the program

eye tracking data format : Index Time Duration X Y Page-link

segmentation format: Index $DiagX_1$ $DiagY_1$ $DiagX_2$ $DiagY_2$ Name

```
#PARAMETERS :-

SegmentationPath =  "C:\\project\\Segmentation"
PageURL =   "https://www.apple.com"
ScanPath =  "C:\\project\\EyeTrackingData"
userList =  [3,4,15,18,21,23,31,32,33,38]
degOfAcc =  0.5
EyeTrackerUserDistance = 60
screenResX =  1366.
screenResY =  768
screenSize =  15.6
```

> Preliminary Stage:

```
AoIs = getAoIs(SegmentationPath)
myErrorRateArea = calcErrRateArea(degOfAcc, EyeTrackerUserDistance, screenResX, screenResY, screenSize)
Participants= getParticipants (userList, ScanPath, PageURL)

sequences = createSequences (Participants, AoIs, myErrorRateArea)

keys = sequences.keys()
for y in range (0 , len (keys)):
    sequences[keys[y]] = sequences[keys[y]].split('.')
    del sequences[keys[y]][len(sequences[keys[y]])-1]
for y in range (0 , len (keys)):
    for z in range (0, len(sequences[keys[y]])):
        sequences[keys[y]][z] = sequences[keys[y]][z].split('-')
```

```
def getParticipants (userList, Path, pageName):
    Users = {}
    for x in userList:
        fileOpen = open (Path + "P" + str(x) + ".txt", "r")
        File = fileOpen.read()
        Records = File.split(newline)
        myRecords_templist = []
        for y in range (1, len( Records) - 1):
            try:
                if  Records[y].index(pageName) > 0:
                    myRecords_templist.append( Records[y].split('\t')) ## list of list
            except:
                continue
        key = "P"
        if x <= 9:
            key += "0"
        Users[key + str(x)] = myRecords_templist
    return Users
```

```
def createSequences (Users, AoIs, errorRateArea):
    Sequences = {}
    keys = Users.keys()
    for y in range (0 , len (keys)):
        sequence = ""
        for z in range (0, len (Users[keys[y]])):
            tempAoI = ""
            tempDuration = 0
            for k in range (0, len (AoIs)):
                if float(Users[keys[y]][z][3]) >= (float (AoIs[k][1]) - errorRateArea) and
                        float(Users[keys[y]][z][3]) < (((float (AoIs[k][1]) - errorRateArea)+(float(AoIs[k][2])+ 2*errorRateArea))) and
                        float(Users[keys[y]][z][4]) >= (float (AoIs[k][3]) - errorRateArea) and
                        float(Users[keys[y]][z][4]) < (((float (AoIs[k][3]) - errorRateArea)+(float(AoIs[k][4])+ 2*errorRateArea))) :
                    tempAoI = tempAoI + AoIs[k][5]
                    tempDuration = int (Users[keys[y]][z][2])
            distanceList = []
            if len (tempAoI) > 1:
                for m in range (0 , len(tempAoI)):
                    for n in range (0, len (AoIs)):
                        if tempAoI[m] == AoIs[n][5]:
                            distance = []
                            for s in range (int (AoIs[n][1]), int (AoIs[n][1]) + int (AoIs[n][2])):
                                for f in range (int (AoIs[n][3]), int (AoIs[n][3]) + int (AoIs[n][4])):
                                    distance.append(sqrt(pow(float(Users[keys[y]][z][3]) - s, 2) + pow(float(Users[keys[y]][z][4]) - f, 2)))
                            distanceList.append([AoIs[n][5], min(distance)])
                distanceList.sort( key=lambda x: x[1])
                tempAoI = distanceList[0][0]
            if len (tempAoI) != 0:
                sequence = sequence + tempAoI + "-" + str (tempDuration) + "."
        Sequences [keys[y]] =  sequence
    return Sequences
```

```
def getAoIs (Path):
    AoIs = []
    fileOpen = open (Path + ".txt", "r")
    File = fileOpen.read()
    Segments = File.split(newline)
    for x in range (0, len (Segments)):
        temp = Segments[x].split(space)
        AoIs.append ([temp[0], temp[1], temp[2], temp[3], temp[4], temp[5]])
    return AoIs
```

The “getAoIs” function takes in the file path to the location containing the segmentation data and returns a list of lists, where each list contains the information of each segments like its number, co-ordinates.

The error rate area is found by passing the degree of the accuracy, the resolution and size of the screen.

The “getParticipants” function takes in arguments of a user list, the path to the folder containing scan-paths of each user and the link to the file on which the eye tracking data is collected. The function returns a “Users” dictionary where each key (participant) gives list of all fixation and its details about the position, duration etc.

The “createSequence” function takes in the User dictionary, list of AoIs, and the errorRateArea as arguments to determine into which AoI each User fixation falls into. The Fixations which are very clearly in the limits of a visual segment is directly appended into the list (the scan-path of the user in terms of visual segment it falls on), the other cases which are ambiguous with regards to their association with a visual element, the distances to all visual elements are calculated and the nearest visual element is associated with the fixation and then appended into the list(scanpath). These lists are individually combined to form the dictionary of scanpath “Sequences” where each user is the key.

> First Pass:

```
Sequences_num = {}
keys = sequences.keys()
for y in range (0 , len (keys)):
    if (len(sequences[keys[y]])!=0):
        Sequences_num[keys[y]] = getNumberedSequence(sequences[keys[y]])
    else:
        Sequences_num[keys[y]] = []

ImportanceThreshold = calculateImportanceThreshold(Sequences_num)
ImportantAoIs = updateAoIsFlag(getNumberDurationOfAoIs(Sequences_num), ImportanceThreshold)
NewSequences = removeInsignificantAoIs(Sequences_num, ImportantAoIs)
```

```
def getNumberedSequence (Sequence):
    numberedSequence = []
    numberedSequence.append([Sequence[0][0], 1, Sequence[0][1]])
    for y in range (1, len(Sequence)):
        if Sequence[y][0] == Sequence[y-1][0]:
            numberedSequence.append([Sequence[y][0], numberedSequence[len(numberedSequence)-1][1], Sequence[y][1]] )
        else:
            numberedSequence.append([Sequence[y][0], getSequenceNumber(Sequence[0:y], Sequence[y][0]), Sequence[y][1]])
    AoIList = getExistingAoIListForSequence(numberedSequence)
    AoINames = getAoIs(SegmentationPath)
    AoINames = [w[5] for w in AoINames]
    newSequence = []
    myList = []
    myDictionary = {}
    replacementList = []
    for x in range (0, len (AoIList)):
        totalDuration = 0
        for y in range (0, len(numberedSequence)):
            if numberedSequence[y][0:2] == AoIList[x]:
                totalDuration = totalDuration + int (numberedSequence[y][2])
        myList.append([AoIList[x], totalDuration])
    for x in range (0, len (AoINames)):
        myAoIList = [w for w in myList if w[0][0] == AoINames[x]]
        myAoIList.sort( key=Lambda x: x[1])
        myAoIList.reverse()
        if len (myAoIList) > 0:
            myDictionary [AoINames[x]] = myAoIList
    for AoI in AoIList:
        index = [w[0] for w in myDictionary[AoI[0]]].index(AoI)
        replacementList.append ([AoI, [AoI[0], (index + 1)]])
    for x in range (0, len(numberedSequence)):
        myReplacementList = [w[0] for w in replacementList]
        index = myReplacementList.index(numberedSequence[x][0:2])
        newSequence.append([replacementList[index][1][0]] + [replacementList[index][1][1]] + [numberedSequence[x][2]])
    return newSequence
```

```
def updateAoIsFlag(AoIs, threshold):
    for AoI in AoIs:
        if AoI [1] >= threshold[0] and AoI [2] >= threshold[1]:
            AoI [3] = True
    return AoIs

def     (Sequences, AoIList):
    significantAoIs = []
    for AoI in AoIList:
        if AoI [3] == True:
            significantAoIs.append(AoI[0])

    keys = Sequences.keys()
    for y in range (0 , len (keys)):
        temp = []
        for k in range (0, len(Sequences[keys[y]])):
            try:
                significantAoIs.index(Sequences[keys[y]][k][0:2])
                temp.append(Sequences[keys[y]][k])
            except:
                continue
        Sequences[keys[y]] = temp
    return Sequences
```

```
def calculateImportanceThreshold (sequences):
    myAoICounter = getNumberDurationOfAoIs(sequences)
    commonAoIs = []
    for myAoIdetail in myAoICounter:
        if myAoIdetail[3] == True:
            commonAoIs.append(myAoIdetail)
    if len (commonAoIs) == 0:
        print "No shared instances!"
        exit(1)
    minValueDuration = commonAoIs[0][2]
    for AoIdetails in commonAoIs:
        if minValueDuration > AoIdetails[2]:
            minValueDuration = AoIdetails[2]
    minValueCounter = commonAoIs[0][1]
    for AoIdetails in commonAoIs:
        if minValueCounter > AoIdetails[1]:
            minValueCounter = AoIdetails[1]
    return [minValueCounter, minValueDuration]
```

```
def removeInsignificantAoIs(Sequences, AoIList):
    significantAoIs = []
    for AoI in AoIList:
        if AoI [3] == True:
            significantAoIs.append(AoI[0])
    keys = Sequences.keys()
    for y in range (0 , len (keys)):
        temp = []
        for k in range (0, len(Sequences[keys[y]])):
            try:
                significantAoIs.index(Sequences[keys[y]][k][0:2])
                temp.append(Sequences[keys[y]][k])
            except:
                continue
        Sequences[keys[y]] = temp
    return Sequences
```

The “getNumberedSequence” takes in an entry from the dictionary returned by the “createSequences” function and returns a sequence (list) that is numbered and appended to another dictionary “Sequences_num”

Taking “Sequences_num, a threshold is found out to distinguish what would be considered as a majority element. This function returns the threshold of the frequency and the duration. Using this information, on iterating over all scanpaths, each AoI is flagged whether if it’s a major/essential element that must be considered for the trending scanpath with the help of the “updateAoIsFlag” function.

The information of the important AoIs along with the numbered sequences are passed as arguments to the “removeInsignificantAoIs” function where as the name suggests, only the important AoIs flagged by the above function are considered to form the new set of Scanpaths.

> Second Pass:

```
NewListAoIs = getExistingAoIList(NewSequences)
NewAoIList = calculateTotalNumberDurationofFixations(NewListAoIs , calculateNumberDurationOfFixations(NewSequences))
FinalList = getValueableAoIs(NewAoIList)
FinalList.sort( key = lambda x: (x[4], x[3], x[2]))
FinalList.reverse()
```

```
def getExistingAoIList (Sequences):
    AoIlist = []
    keys = Sequences.keys()
    for y in range (0, len(keys)):
        for x in range (0, len(Sequences[keys[y]])):
            try:
                AoIlist.index(Sequences[keys[y]][x][0:2])
            except:
                AoIlist.append(Sequences[keys[y]][x][0:2])
    return AoIlist
```

```
def getValueableAoIs (AoIList):
    commonAoIs = []
    valuableAoIs = []
    for myAoIdetail in AoIList:
        if myAoIdetail[5] == True:
            commonAoIs.append(myAoIdetail)

    minValue = commonAoIs[0][4]
    for AoIdetails in commonAoIs:
        if minValue > AoIdetails[4]:
            minValue = AoIdetails[4]

    for myAoIdetail in AoIList:
        if myAoIdetail [4] >= minValue:
            valuableAoIs.append(myAoIdetail)

    return valuableAoIs
```

>Printing the Trending Scan-path:

```
TrendingSequence = []
for y in range (0, len(FinalList)):
    TrendingSequence.append(FinalList[y][0])
print "Trending Scanpath Sequence is:", getAbstractedSequence(TrendingSequence)
```

A new list of all the important AoIs is made using the “getExistingAoIList” function. The new list of AoIs is passed through the “getValuableAoIs” to final list of AoIs that make up the Trending Scanpath. These AoIs are sorted in the order that they occur in the most. We now have the final list of AoIs that form the Trending Scanpath and is now printed.

Conclusions on STA

To summarise the whole STA algorithm, we have it, being given, the eye tracking data and the segmented webpage visual elements as input. In the "Preliminary stage" individual scanpath are created considering the visual elements of a webpage. This is then passed to the next stage-"First pass", where the trending elements are identified. The trending scanpath is constructed using these trending elements based on their overall location in the individual scanpaths in the "Second Pass".

Eye tracking studies done on the STA algorithm gave positive results showing STA algorithm's capabilities to generate a trending scanpath that had all the trending visual elements in the proper order without missing out on any elements that were visited by all or atleast those that atleast got similar attention. At the same time it also linked each of the visual elements with its correlating part in the source code of the page using the help of the VIPS algorithm.

Be that as it may, STA has its own set of limitations inspite of the huge number of other issues it has solved. So far, the STA algorithm has been used with one segmentation technique and on a set granularity level. The performance of STA algorithm with a change in these parameter is still not known. During the generation of the trending scanpath, we may observe some accuracy problems with the fixations' positions and have to decide to which visual element a

fixation belongs to when it is overlapping with more than one visual element. The number of people required for a user study has been debated a lot, this also causes a with the evaluation of STA, since the sample size of users wasn't very big. The STA algorithm results are still unknowm w.r.t users belonging to different regions of world, race, level of literacy etc. While at the same time, the STA algorithm is also constrained with the slowest part of the algorithm having a time complexity of $O(n^6)$. An area to work on STA would be developing a quicker and quite efficient method to cut short the time taken to decide the visual element to which a shared fixation belongs. This seems probable, which makes it necessary to conduct a study to find a solution/method as it significantly reduces the algorithm's time complexity.

Work done in the project

At present as a part of this project, the basic implementation of the STA is completed and was written in Python 3 while taking the help of the code shared by Eraslan et.al(2016) as implementation of certain functions of the algorithm were unclear. Due to lack of access to sufficient user eye tracking data in the required format, time was spent on creating a data set to test the original code. With the help of a colleague, we used the maps we found with the User's fixation various websites on them and had to convert them to the format as required. Since the results were difficult to visualize in the data form, further analysis was difficult.

The ability of STA to generate a trending scanpath over a website for improving the accessibility of the website for visually disabled users as well as knowledge to improve the design of a website is limited by its time complexity and also to those developers with access to multiple users at the place of development of that website to collect their eye tracking data. Since the need to make a website with good accessibility is important, it brings up the need to create a solution to provide the capabilities to wider amount of developers. An improvement towards the time complexity of the Algorithm remains the main priority of the project right now. After the understanding the implementation of the STA algorithm, I have worked on trying to make improvements at the function level of the STA algorithm.

Before the preliminary phase, while the segmentation of the of each page happens, instead of passing the co-ordinates of the segment in a text file, creating a 2-D matrix where in each cell is a pixel inside which the information regarding what visual segment it belongs to is stored. Thus on iterating through the fixations of a user, we can directly use the co-ordinates of the fixation as the index to get the information about which segment it falls on. This approach of using a matrix may prove to space wise heavy, but is definitely an option to be considered against the huge time complexity at that step. Progress is being made in this front to create the segmentation data in a 2D matrix and to adapt the code for the change in the input format.

In all, i have tried to implement the STA algorithm as the authors have implemented to get an idea about the inner details of the algorithm to find areas which can be worked upon to increase the speed or reduce complexity

References

1 Eraslan, Sukru, Yeliz Yesilada, and Simon Harper. "Scanpath trend analysis on web pages: Clustering eye tracking scanpaths." *ACM Transactions on the Web (TWEB)* 10, no. 4 (2016): 1-35.