**Universidade Federal do ABC**
**Programa de Pós-graduação em Ciência da Computação**

**Inteligência na Web e Big Data**

# Projeto - Implementação de um algoritmo para análise de Logs de Eye-Tracking

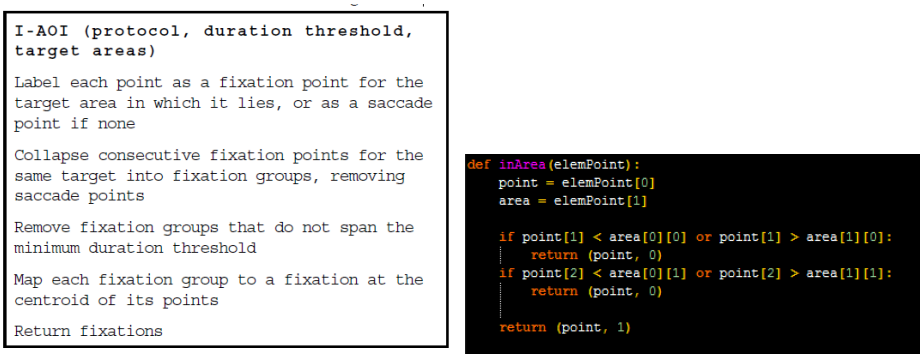
**Thiago Donizetti dos Santos**
*Universidade Federal do ABC*
*thiago.donizetti@aluno.ufabc.edu.br*

## 1. Introdução

Eye-tracking é uma técnica na qual utiliza-se câmeras específicas para capturar o movimento do olho do usuário durante sua interação com o computador ou outros artefatos. O uso desta técnica é empregado em diferentes áreas e para diversos fins, entre eles estão estudos de usabilidade na web [2], saúde [1], direção de veículos e leitura [3].

De acordo com Salvucci e Goldberg a analise dos movimentos dos olhos é feita de acordo com Fixações e Sacadas, dado que a primeira é definida como sendo o momento em que os olhos estão parados sobre alguma região de interesse e a segunda como movimentos rápidos entre as fixações. Entre as informações analisadas estão a duração das fixações, velocidade das sacadas, regiões de interesse, e outras. Neste contexto, a análise de logs gerados por Eye-tracking requer a identificação de fixações, localizando em que áreas ocorreram e em que momento. Na literatura é possível encontrar diferentes algoritmos e métodos utilizados no problema de identificação de fixações e este trabalho é baseado no artigo ”Identifying Fixations and Saccades in Eye-Tracking Protocols” [3], no qual os autores apresentam algoritmos de identificação em relação à velocidade, dispersão e área.

Baseado neste artigo e na disciplina de Inteligência na Web e Big Data, este trabalho apresenta a implementação do algoritmo ”Area-of-Interest Identification (I-AOI)”, um algoritmo baseado em área, utilizando técnicas de paralelização e distribuição.




(a) Pseudocódigo do algoritmo I-AOI (b) Subrotina inArea

Figura 1: Figura

## 2. Algoritmo

O algoritmo I-AOI é utilizado para identificar as fixações que ocorrem em áreas específicas, chamadas aqui de *target areas*. As *target areas* são regiões retangulares de interesse que representam áreas com informação no campo de visão, como imagens e botões em um site, por exemplo.

A fixação é definida então por momentos em que o olhar, identificado por pontos em duas dimensões, se encontra dentro de uma das áreas definidas somado à um *threshold* de duração para distinguir fixação de passadas rápidas pela área.

Inicialmente o I-AOI rotula todos os pontos nos dados que estão dentro de uma das *target areas* como fixações e os que estão fora como sacadas.

No próximo passo, os pontos de fixação consecutivos que estão nas mesmas áreas são colapsados em grupos de fixação descartando os pontos de sacada.

Os grupos então são filtrados retirando aqueles que estiverem abaixo do *threshold* de duração definido e são então transformados em tuplas.

O pseudocódigo do I-AOI é apresentado na Figura 1(a).

## 3. Implementação

A implementação do algoritmo foi feito utilizando a linguagem Python bem como a API de Spark para Python (PYSPARK).

O algoritmo foi divido em duas partes: Rotina principal e uma subrotina:

### *3.1. Subrotina*

Método nomeado inArea, recebe as coordenadas de um ponto e da área. Os pontos são representados por coordenadas X e Y e um timestamp que indica o momento do evento. Cada área é definida por duas tuplas, onde a primeira contém as coordenadas do ponto inicial da área e a segunda as coordenadas do ponto final, delimitando assim, a área.

Esa subrotina devolve uma tupla contendo o ponto e 1 caso o ponto esteja naquela área ou 0, caso não pertença. Exemplo. ( (1234, X, Y), 0) seria um ponto (X,y) que não esteja na área, ou seja, é um ponto de sacada em relação a ela. ( (2341, Z, W), 1 ), neste outro exemplo, o ponto (Z,W) se encontra na região da área e foi marcado com 1. Foi optado por rotular com 0 ou 1 para que fosse possível um passo de redução da rotina principal, na qual os rótulos das tuplas com a mesma chave, sendo a chave igual a (timestamp, X, Y), são somados. Neste passo, caso a soma para uma chave seja igual a 1 o ponto teve fixação em uma área, se foi 0 não teve fixação em nenhuma área. A implentação é exibida na Figura 1(b).

### *3.2. Rotina principal*

chamada de ”AreaOfInterestParalelo”, recebe como entrada os dados do log com os pontos e uma lista de áreas.

O primeiro passo do algoritmo é criar as RDDs para os pontos e para as áreas. Após isso um produto cartesiano é aplicado para combinar todos os pontos com todas as áreas, conforme mostrado na figura 2(a).

```
def AreaOfInterestParalelo(dataset, areas):

    dataRDD = sc.parallelize(dataset, 4)
    areasRDD = sc.parallelize(areas,4)
    cartesianAreasData = dataRDD.cartesian(areasRDD)
```

(a) Criando as RDDs e produto cartesiano

```
labeledPoints = cartesianAreasData.map(inArea).reduceByKey(add)
```

(b) Map e reduce

Figura 2: Cartesiano, Map e reduce

O RDD resultante é então mapeado para que cada ponto seja rotulado como sendo fixação ou sacada para cada uma das áreas. Após o mapeamento, uma redução é feita para identificar pontos que tiveram fixação para alguma das áreas, conforme mencionado anteriormente e mostrado na figura 2(b).

Os dados precisam agora ser ordenados para que o próximo passo seja coerente. Nele, as fixações consecutivas devem ser agrupadas, por isso a ordem é importante. O loop for separa em grupos as fixações consecutivas conforme a figura 3 mostra.

Esta lista de grupos é então paralelizada para um RDD. Com ela, duas outras são criadas: Uma com a quantidade de pontos em cada grupo e outra com a soma dos pontos em cada grupo. Figura 4.

As RDDs então são unidas e um filtro é aplicado retirando os grupos que não atendem ao threshold de duração estabelecido previamente (Figura 5(a)).

```
orderedData = labeledPoints.sortBy(lambda x: x[0][0]).collect()

FixationGroups = []
saccadePoint = False

count = -1
for tupla in orderedData:
    if tupla[1] == 0:
        saccadePoint = True
    else:
        if len(FixationGroups) == 0 or saccadePoint:
            count += 1
            saccadePoint = False
        FixationGroups.append((count,tupla[0]))

fixationsRDD = sc.parallelize(FixationGroups, 4)
```

Figura 3: Separação dos grupos

```
fixationsRDD = sc.parallelize(FixationGroups, 4)

countPoints = sc.parallelize(FixationGroups, 4).map(lambda x: (x[0], 1)).reduceByKey(add)
somaCoordTime = fixationsRDD.reduceByKey(lambda (tsa, xa, ya),(tsb, xb, yb): ((tsa+tsb, xa+xb, ya+yb)))
```

Figura 4: Soma e contagem

No próximo passo, a média dos grupos restantes são calculados para que sejam obtidos os centroides de cada um deles.

O algoritmo então exibe na tela os centroides resultantes como pode ser visto na Figura 5(b).

```
zippedCentroids = somaCoordTime.join(countPoints)
filtredDurations = zippedCentroids.filter(lambda (x,y): y[1] > TIMETHRESHOLD)
```

(a) União das RDDs e threshold

```
centroids = filtredDurations.map(lambda (k,v): (k, v[0][0]/v[1], v[0][1]/v[1], v[0][2]/v[1] ))
print centroids.collect()
```

(b) Centroides e resultado

Figura 5: Threshold, médias e resultado final

## 4. Resultados

Implementando o algoritmo com PySpark foi possível efetuar quase todas as operações necessárias apenas usando os comandos básicos de map, reduce e filter. A única parte que não foi possível utilizar tais comandos foi a parte em que é executado um laço para separar os grupos, uma vez que um grupo de fixações deve acabar e um novo deve começar sempre que é encontrado um ponto de sacada. Como a iteração depende do ponto seguinte, não foram encontradas maneiras alternativas a não ser o loop utilizando o for.

## 5. Conclusão

Os resultados foram satisfatórios uma vez que a implementação apresentada soluciona o problema de forma simples e elegante. Estudos mais avançados sobre o tema serão necessários para que seja possível a execução de todas as tarefas sem o uso de estruturas de repetição como o comando for utilizado.

## 6. Referências


[1] Zillah Boraston and Sarah-Jayne Blakemore. The application of eye-tracking technology in the study of autism. *The Journal of physiology*, 581(3):893–898, 2007.

[2] Jakob Nielsen and Kara Pernice. *Eyetracking web usability*. New Riders, 2010.

[3] Dario D Salvucci and Joseph H Goldberg. Identifying fixations and saccades in eye-tracking protocols. In *Proceedings of the 2000 symposium on Eye tracking research & applications*, pages 71–78. ACM, 2000.