UNIVERSIDADE FEDERAL DE SANTA CATARINA

Análise da Redução de Dados em Trajetórias de Objetos Móveis

Hércules Mateus Avancini

Orientadora : Vania Bogorny

Florianópolis – SC

2010/2

UNIVERSIDADE FEDERAL DE SANTA CATARINA – UFSC
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA – INE
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

Análise da Redução de Dados em Trajetórias de Objetos Móveis

Hércules Mateus Avancini

Trabalho de Conclusão de Curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação.

Florianópolis – SC
2010/2

Hércules Mateus Avancini

Análise da Redução de Dados em Trajetórias de Objetos Móveis

Trabalho de Conclusão de Curso apresentado como parte
dos requisitos para obtenção do grau de Bacharel
em Sistemas de Informação

Orientadora:

_____

– Profa. Vania Bogorny

Banca Examinadora:

_____

– Prof. Ronaldo dos Santos Mello

_____

– Profa. Patrícia Della Mea Plentz

AGRADECIMENTOS

Agradeço primeiramente a Deus por poder viver experiências boas e ruins da minha vida no período do curso de graduação de sistemas de informação da Universidade Federal de Santa Catarina.

Dedico este trabalho à minha família que sempre me incentivou e apoiou nos momentos mais difíceis desta caminhada. Em especial meu pai João Avancini, minha mãe Osmarina Blasius, minha madrasta Jucélia Carmem Mota Avancini e minha irmã Patrícia Carmem Rodrigues que nunca me privaram da oportunidade de estudar, que sempre me apoiaram em momentos de desanimo e que são motivos de orgulho em minha vida.

Agradeço ao meu namorado Marcos Antônio Custódio pelo amor, carinho, paciência que soube ser compreensivo em momentos de prova, trabalho e principalmente quando eu precisava de uma orientação.

Aos meus amigos de classe, em especial Carlene Maria Schlemper e Orlinda Lúcia de Souza Alosilla, que compartilhamos alegrias e tristezas, provas e trabalhos, e claro a vitória da conclusão deste curso.

Agradeço à professora e orientadora Vania Bogorny pelo desafio deste trabalho, dedicação e amizade dispensada. Aos professores Ronaldo dos Santos Mello e Patrícia Della Mea Plentz que aceitaram participar da banca de trabalho e que contribuiu de forma significativa para a realização do mesmo.

Enfim, agradeço a todos que me auxiliaram nesta caminhada tão importante para atingir essa grande etapa da minha vida.

RESUMO

Com a redução dos preços de aparelhos de coleta de informações geográficas aumentou significativamente a disponibilidade de bases de dados geográficas. Essas bases de dados são formadas por conjuntos de trajetórias brutas de objetos móveis. No entanto, essas trajetórias não possuem contexto ou semântica associada. Por isso foram criados alguns algoritmos para adicionar semântica a essas trajetórias. No entanto, os tempos de processamento desses algoritmos são elevados, podendo chegar a mais de 6 horas dependendo do volume de dados. Para isso, este trabalho vem com o objetivo de otimização deste tempo de processamento, através da aplicação de técnicas de amostragem estatística para reduzir o volume de dados das trajetórias brutas antes da aplicação do algoritmo, reduzindo assim o tempo de processamento. Técnicas foram aplicadas sobre dois tipos de trajetórias diferentes e, posteriormente, foi analisado o tempo de processamento que se consegue otimizar e se a redução de dados afeta os resultados do algoritmo.

Palavras-chave: bases de dados geográficas, trajetórias brutas, objetos móveis, tempo de processamento, técnicas de amostragem estatística, redução de dados.

ABSTRACT

With the decreasing price of equipments for collecting geographic information, the availability of geographic databases increased significantly. These databases store what we call raw trajectories of moving objects. However, these trajectories have no context or semantics associated to the original data. Therefore, some algorithms have been developed to add semantic information to trajectories. The processing time of these algorithms, however, is high, reaching more than six hours depending on the size of the database. To reduce this problem, this work aims at optimizating processing time through the application of statistical sampling techniques to reduce the volume of raw data of trajectories before running the algorithms. We applied these techniques on two different types of trajectories and, subsequently, we evaluated the processing time and the quality of the output data.

Key-words: *geographical databases, raw trajectories, moving objects, processing time, statistical sampling techniques, data reduction.*

SUMÁRIO

LISTAGEM DE TABELAS **8**

LISTAGEM DE FIGURAS **9**

LISTAGEM DE GRÁFICOS **10**

1. INTRODUÇÃO **11**

2. CONCEITOS BÁSICOS **14**

2.1. TRAJETÓRIAS 14
2.2. TRAJETÓRIAS SEMÂNTICAS 15
2.3 O ALGORITMO CB-SMOT 18

3. MÉTODOS PARA REDUÇÃO DE DADOS **20**

3.1. ELIMINAÇÃO DE RUÍDO 21
3.2 AMOSTRAGEM ALEATÓRIA SIMPLES 23
3.3. AMOSTRAGEM SISTEMÁTICA 23
3.4. IMPLEMENTAÇÃO 24

4. EXPERIMENTOS E ANÁLISE DOS RESULTADOS **27**

4.1. CONJUNTO DE TRAJETÓRIAS DE CARROS 27
4.1.1. Análise do tempo de processamento *27*
4.1.2. Análise do total de stops gerados *29*
4.2. CONJUNTO DE TRAJETÓRIAS DE PEDESTRES 35
4.2.1. Análise do tempo de processamento *36*
4.2.2. Análise do total de stops gerados *38*
4.3 DISCUSSÃO DOS RESULTADOS 47

5. CONCLUSÃO E TRABALHOS FUTUROS **49**

6. REFERÊNCIAS **51**

ANEXO 1 – ARTIGO. **53**

ANEXO 2 – CÓDIGO FONTE. **67**

LISTAGEM DE TABELAS

Tabela 1 - Exemplo de uma trajetória armazenada em uma tabela de banco de dados... 12
Tabela 2 - Porcentagem de otimização do tempo de processamento do algoritmo CB-SMOT 29
Tabela 3 - Porcentagem de otimização do algoritmo CB-SMOT conforme a retira de pontos. 38

LISTAGEM DE FIGURAS

Figura 1 - Uma trajetória da cidade do Rio de Janeiro. 12
Figura 2 - Trajetória bruta representado por diversos pontos no espaço. 15
Figura 3 - Trajetória semântica, focando os movimentos e os pontos de parada de um turista 16
Figura 4– Exemplo de trajetória com 2 stops e 2 unknown stops (figura obtida em [Palma 2008]). 19
Figura 5 – Exemplo de ponto gerado errado 22
Figura 6 – a) Tela de seleção de trajetórias. b) Tela de limpeza de trajetórias..... 25
Figura 7 – Diagrama das classes das telas alteradas ou inseridas. a) Atributos e métodos adicionados à tela Trajectory. b) Tela “Cleanning Trajectory” adicionada ao modulo STPM. 25
Figura 8 – Exemplo de limpeza com remoção de até 4 pontos. 31
Figura 9 – Exemplo de limpeza e execução do algoritmo CB-SMOT onde os stops originais são em maior número que os stops das trajetórias limpas..... 32
Figura 10 – Exemplo de limpeza e execução do algoritmo CB-SMOT na qual os stops originais foram divididos em dois os stops nas trajetórias limpas. 33
Figura 11 – Exemplo de limpeza com remoção de até 4 pontos em uma trajetória. 39
Figura 12 – Exemplo de limpeza e execução do algoritmo CB-SMOT onde os stops originais são em maior número que os stops das trajetórias limpas. 40
Figura 13 – Exemplo de limpeza e execução do algoritmo CB-SMOT onde os stops originais são diferentes dos stops das trajetórias limpas. 45
Figura 14 – Exemplo do cálculo da velocidade média para trajetória original e limpa. . 48

LISTAGEM DE GRÁFICOS

Gráfico 1 - Redução do tempo de processamento das trajetórias de carros. 29
Gráfico 2 - Quantidade de stops por tabela. 30
Gráfico 3 - Gráficos de velocidade pelo tempo de cada ponto da trajetória 787, com stops ou candidatos a stop das trajetórias original e limpas. 35
Gráfico 4 - Redução do tempo de processamento para trajetórias de pedestres..... 37
Gráfico 5 - Quantidade de stops por tabelas. 39
Gráfico 6 – Gráficos de velocidade onde mostra que a execução do algoritmo CB-SMOT criou mais stops e em locais diferentes. 43
Gráfico 7 – Gráficos de velocidade onde mostra que a execução do algoritmo CB-SMOT criou mais e menos stops em locais diferentes. 47

1. INTRODUÇÃO

A redução nos preços de aparelhos de coleta de informações geográficas como aparelhos de Sistema de Posicionamento Global (GPS), de telefone celular ou qualquer outro dispositivo móvel aumentou significativamente a utilização de serviços de georeferenciamento por todos os segmentos da população. Inúmeras bases de dados geográficas estão sendo geradas com grandes volumes de dados. Essas bases são formadas por conjuntos de percursos percorridos por um corpo em um determinado período de tempo. Este percurso pode ser chamado de trajetória.

Uma trajetória consiste no caminho percorrido por um dado objeto no espaço durante um movimento. Para o escopo deste trabalho, uma trajetória é um conjunto de pontos no espaço (x,y) ordenados no tempo (t). Ou seja, podemos definir uma trajetória como um conjunto de pontos (x,y,t) .

Com isso, podemos definir que um objeto móvel é qualquer objeto cuja posição geográfica muda continuamente com o passar do tempo, como por exemplo quando uma pessoa caminha, mesmo que em distâncias curtas, em um shopping center. As trajetórias desses objetos móveis atualmente estão sendo utilizadas em análises de uma série de problemas atuais e relevantes, tais como o caos do transporte nos grandes centros, desastres climáticos, monitoramento de criminosos e a transmissão de doenças através da migração de aves.

Os aparelhos de GPS são exemplos de dispositivos que geram a referência espacial de um objeto no decorrer do tempo. A maioria destes dispositivos geram grandes quantidades de dados com informações redundantes, uma vez que por *default* coletam um ponto a cada segundo para a trajetória. Assim, quanto menor o intervalo de tempo de coleta, entre dois pontos, maiores serão as trajetórias e as bases de dados.

A Figura 1 ilustra um exemplo de apenas uma trajetória projetada na ferramenta Quantum Gis [Quantum 2010].

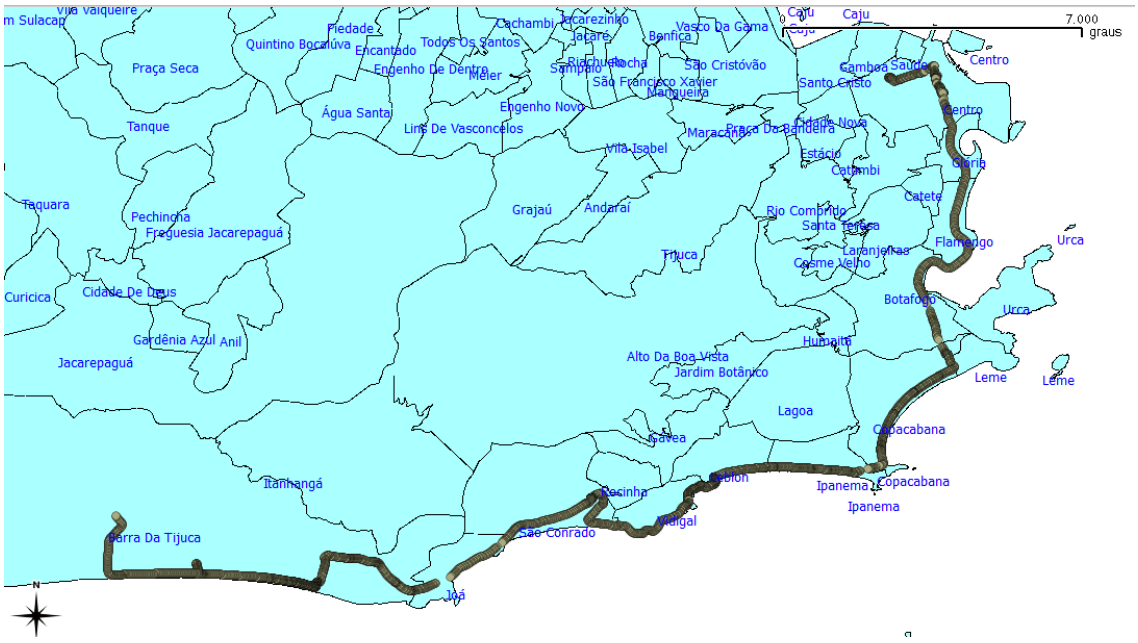


Figura 1 - Uma trajetória da cidade do Rio de Janeiro.

A Figura 1 mostra uma trajetória de apenas uma hora e meia de duração totalizando 4.907 (quatro mil novecentos e sete) registros que vão do bairro Barra da Tijuca até o Centro da cidade do Rio de Janeiro. A mesma trajetória pode ser observada na forma de registros em um banco de dados. A coluna *gid* representa o atributo identificador seqüencial de cada ponto. A coluna *tid* representa a coluna que identifica qual trajetória o ponto pertence. A coluna *time* representa a tempo em que o ponto foi armazenado, e a coluna *the_geom* representa a coordenada geográfica de cada ponto, ou seja, define a localização do ponto no espaço.

Gid	Tid	Time	The_Geom
12393	779	"2007-03-05 16:29:09"	"0101000000D675160591ED2441128A7F1A0F7B5C41"
12394	779	"2007-03-05 16:29:10"	"01010000006842F99A99ED2441644FA1750F7B5C41"
...	...	....	...
17298	779	"2007-03-05 17:58:31"	"010100000018818A584F602441137DA8A906715C41"
17299	779	"2007-03-05 17:58:32"	"01010000006A39A002366024417CD767F909715C41"

Tabela 1 - Exemplo de uma trajetória armazenada em uma tabela de banco de dados.

Conforme ilustrado na tabela 1, os dados da trajetória são armazenados no banco de dados geográfico na formal vertical, isto é, cada ponto da trajetória representa um registro no banco de dados. Isto é necessário porque um banco de dados geográfico não suporta pontos móveis, ou seja, para atribuir um tempo a cada ponto é necessário criar

um atributo para representar o espaço e outro para representar o tempo, gerando assim, uma tupla para cada ponto com seu respectivo tempo.

As trajetórias brutas na forma como são criadas contém poucas informações relevantes, basicamente representando pontos no tempo e no espaço. Por isso, trabalhos acadêmicos relativos à análise e o enriquecimento semântico destas trajetórias vêm sendo desenvolvidos com o intuito de descobrir conhecimento sobre toda ou parte das mesmas.

Recentemente, Spaccapietra [Spaccapietra 2008] introduziu um novo modelo de raciocínio sobre trajetórias, que nos permite uma poderosa análise semântica, baseado nos conceitos de *stops* e *moves*. O primeiro é um local de parada do objeto por um determinado período de tempo. Já o segundo representa os pontos da subtrajetória que estão entre os *stops*.

Todos os *stops* precisam ser calculados, e o método varia de acordo com a aplicação e o interesse do usuário. Nosso grupo de pesquisa já desenvolveu três métodos para este fim, IB-SMOT [Alvares 2007], CB-SMOT [Palma2008] e DB-SMOT [Manso2010]. O primeiro calcula *stops* com base na intersecção da trajetória com objetos geográficos de interesse. O segundo calcula *stops* em lugares onde a velocidade do objeto é baixa e o terceiro gera *stops* onde a direção da trajetória varia mais que um determinado valor. Para todos os métodos, os *stops* são gerados quando o objeto permaneceu no *stop* por um período mínimo de tempo.

Considerando uma amostra de dados real de veículos que circulam na cidade do Rio de Janeiro, uma tabela de trajetórias do banco de dados tem 7 milhões de pontos e 2 mil trajetórias, podendo levar mais de 6 horas para realizar uma análise de enriquecimento semântico destas trajetórias para gerar os *stops* e *moves*. Com isso, vê-se a necessidade da otimização destes algoritmos ou a redução do volume de dados para a diminuição do tempo de processamento.

O principal objetivo deste trabalho é diminuir consideravelmente o tempo de processamento destes algoritmos de enriquecimento semântico de trajetórias. Para isto será efetuada, antes da execução do algoritmo, uma limpeza nos dados de trajetórias através de algumas técnicas de redução de dados e amostragens.

Nada foi encontrado sobre técnicas para a redução de dados de trajetórias de objetos móveis. No entanto, um trabalho que mais se aproxima dos conceitos de limpeza de trajetória é o trabalho de Pelekis [Pelekis 2008], que fala sobre a reconstrução de trajetórias a partir de seqüências brutas de pontos espaciais, coletados em um

determinado período, para utilização em Data Warehouses de Trajetórias (TDW). Neste processo Pelekis encontra e nomeia várias trajetórias que possuem dentre esse conjunto de pontos espaço-temporais, utilizando alguns parâmetros que são similares as técnicas sugeridas por esse trabalho no capitulo 3.

O restante deste trabalho é organizado da seguinte forma: O Capítulo 2 apresenta alguns conceitos de trajetórias brutas e semânticas. No Capítulo 3 são abordados os métodos de redução de dados e uma breve descrição da implementação destes métodos. No Capítulo 4 são apresentados experimentos e finalmente o Capítulo 5 conclui o trabalho e comenta sobre possíveis trabalhos futuros.

2. Conceitos básicos

Para entendermos como serão aplicadas as técnicas de redução e limpeza dos dados de trajetórias de objetos móveis alguns conceitos sobre trajetórias brutas e trajetórias semânticas serão apresentados. Alguns conceitos básicos sobre o algoritmo *CB-SMOT* também serão apresentados, o qual foi utilizado para avaliar os resultados experimentais deste trabalho.

O *CB-SMOT* foi escolhido porque ele é baseado na variação da velocidade média da trajetória e tem por objetivo identificar as partes de uma trajetória que possuem velocidade abaixo de outras regiões da mesma trajetória. Ainda, porque os dados reais utilizados para análise dos experimentos são de trajetórias de carros e de pedestres, onde o objetivo é identificar regiões de velocidade lenta.

2.1. Trajetórias

Como citado anteriormente, trajetórias de objetos móveis podem ser consideradas como o caminho seguido por um objeto em movimento no tempo e no espaço. Para este escopo, uma trajetória é um conjunto de pontos no espaço (x,y) ordenados pelo tempo (t). Cada ponto em uma trajetória representa uma posição no espaço em um determinado instante de tempo.

Nem todo movimento é considerado uma trajetória. Normalmente o objeto móvel se move durante um período de tempo com um objetivo definido e com demarcação de horário de início e fim, para ser considerado um percurso de interesse. Em outras palavras, uma trajetória é o segmento espaço-temporal do caminho percorrido por um objeto [Spaccapietra 2008].

Uma pessoa com qualquer aparelho de georeferenciamento pode se movimentar de várias formas, gerando assim vários tipos de trajetórias diferentes. Por exemplo, a trajetória de uma viagem de avião, de carro ou a pé tem características completamente diferentes. Essas trajetórias somadas em período de tempo formam um grande volume de dados. No entanto, essas trajetórias geralmente são apenas dados armazenados, sem informação adicional sem qualquer contexto ou semântica, sendo posteriormente um tipo de trajetória muito difícil de analisar, entender e extrair informações interessantes. Esse tipo de trajetória é chamado de trajetória bruta.

Definição 1. *Trajetória bruta*: Uma trajetória bruta é uma lista de pontos espaço-temporal $\{ p_0= (x_0, y_0, t_0), p_1= (x_1, y_1, t_1),... p_n=(x_n, y_n, t_n)\}$, onde $x_i,y_i \in \Re$, $t_i \in \Re^+$, para $i = 0,1,2,...,n$ e $t_0<t_1<t_2...<t_n$.

Trajetórias brutas são apenas seqüências de pontos espaciais ordenados pelo tempo de ocorrência de cada ponto. A Figura 2 mostra uma trajetória bruta ao longo do tempo.

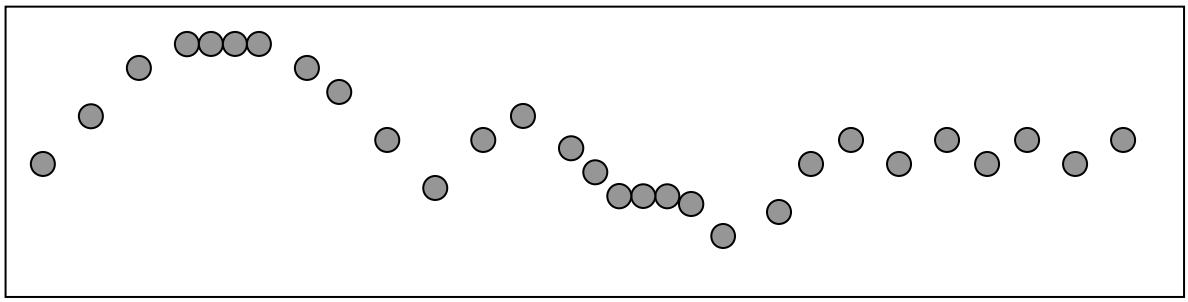


Figura 2 - Trajetória bruta representado por diversos pontos no espaço.

2.2. Trajetórias semânticas

Tendo em vista a desvantagem referente às trajetórias brutas citadas anteriormente, as trajetórias semânticas apresentam a vantagem de serem melhor compreendidas, pois além dos pontos ou dados brutos da trajetória são apresentadas informações adicionais da trajetória ou da região onde ela está inserida. A Figura 3 apresenta a trajetória de um turista enriquecida semanticamente com os lugares que o turista visitou.

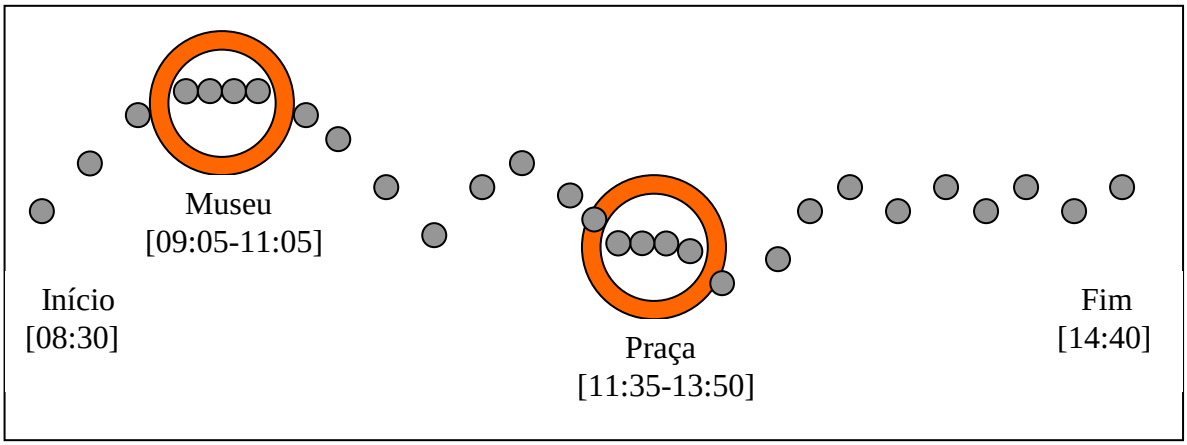


Figura 3 - Trajetória semântica, focando os movimentos e os pontos de parada de um turista

Algumas informações semânticas podem ser retiradas da Figura 3, como o que o turista desta trajetória fez entre as 08:30 da manhã até as 14:40. Já se compararmos várias trajetórias, de vários turistas, pode-se saber, por exemplo, quais pontos turísticos são mais procurados, ou quais conjuntos de pontos que são visitados primeiro. Essas informações podem ser usadas para mineração de dados e obterem-se várias conclusões sobre o contexto das trajetórias analisadas.

As trajetórias semânticas são obtidas através de métodos inteligentes. Alvares [ALVARES 2007] cita que para muitos domínios de aplicação, informações úteis só podem ser extraídas a partir de dados de trajetórias se alguma semântica ou informação geográfica forem considerados. Uma das formas que tem sido utilizada para enriquecer as trajetórias com informações semânticas é através do método stops e moves, proposto por Spaccapietra [Spaccapietra 2008].

Os *stops* representam a parte mais relevante de uma trajetória para uma aplicação, considerando que o objeto tenha permanecido um período mínimo de tempo no mesmo local.

Em uma aplicação de turismo, um *stop* pode ser um ponto turístico, um hotel ou um aeroporto. Em um aplicativo de gerenciamento de tráfego, os lugares importantes (*stops*) podem ser semáforos, rótulas e parques de estacionamento. Dependendo da aplicação e das características dos dados, a duração mínima dessas paradas relevantes pode variar significativamente.

Definição 2: *Candidato a stop*. Um candidato a stop é definido como uma tupla (Rc e Δc) definida pelo usuário, onde Rc é a geometria do objeto geográfico de interesse e Δc é a duração mínima.

Em uma aplicação de turismo, por exemplo, a geometria de um hotel pode ser um candidato e a duração mínima de 3 horas seria o tempo em que uma trajetória deveria interceptar este hotel. Dependendo do candidato a stop, o tempo mínimo pode variar.

O conjunto de candidatos a stop caracteriza os aspectos importantes de uma aplicação.

Definição 3: *Aplicação.* Uma aplicação é um conjunto finito de candidatos a stop $A = \{C_N = (R_{cN} , \Delta_{cN})\}$ onde a geometria de cada candidato a stop não intercepta a geometria do outro. Essa limitação ocorre devido a restrições definidas pelo algoritmo CB_SMOT proposta por [Palma 2008].

Definição 4: *Stop*: Um stop é uma parte da trajetória que representa uma característica importante de um domínio de aplicação e precisa durar um tempo mínimo, podendo este interceptar ou não um candidato a stop.

Diversos métodos já foram citados para gerar stops. Em [Álvares 2007], por exemplo, um stop deve interceptar um candidato a stop por um tempo mínimo. Em [Palma 2008], um stop é uma região de velocidade baixa de uma trajetória por um mínimo período de tempo, podendo esta região interceptar ou não um candidato. Já em [Manso 2010] um stop é uma região da trajetória onde a direção teve uma variação significativa por um período de tempo. Em suma, a definição de stop varia com o método usado para gerá-lo.

Neste trabalho, um stop é definido como proposto por [Palma 2008].

Definição 5: *Move*: um move de uma trajetória T com relação ao conjunto de candidatos a stop é:

(i) a maior subtrajetória contínua de T entre dois stops consecutivos de T, ou;

(ii) a maior subtrajetória contínua de T entre o ponto inicial de T e o primeiro STOP de T; ou;

(iii) a maior subtrajetória contínua de T entre a última parada de T e o último ponto de T, ou;

(iv) a própria trajetória de T, se T não tem stops.

Em outras palavras, cada ponto de uma trajetória que não está no *stop* está em um *move*. Um *move* não tem tempo de duração mínimo e pode ou não cruzar com um

candidato a *stop*. Caso isso aconteça, o intervalo de tempo de interseção deve ser menor do que o tempo mínimo do candidato a stop.

Palma [Palma2008] definiu conceitos para a mineração de trajetórias considerando o contexto de informações geográficas e também as propriedades geométricas das trajetórias, como a velocidade. Esses conceitos são apresentados na próxima seção.

2.3 O algoritmo CB-SMOT

O algoritmo CB-SMOT (Clustering-*Based Stops and Moves of Trajectories*), proposto por [Palma2008], tem duas etapas principais: clusterização e atribuição de semântica. Na parte de clusterização, CB-SMOT foi baseado em um algoritmo de densidade usando o conceito espaço-temporal. CB-SMOT considera adicionalmente o tempo no processo de clusterização, de forma que os clusters correspondem a trechos em que a velocidade é lenta.

No método CB-SMOT, as trajetórias são analisadas com base na sua geografia, no contexto que existe no espaço físico e na sua velocidade. Dessa forma, têm-se dois aspectos envolvidos com a trajetória: o primeiro é relacionado apenas com a trajetória (sua velocidade) e o segundo é o contexto espacial que existe por trás da trajetória (geografia). Cada aspecto é tratado de uma maneira independente. O primeiro utiliza clusterização para definir os trechos de menor velocidade. O segundo aspecto usa os candidatos a stop para atribuir semântica. Este aspecto relaciona a trajetória com um contexto que depende dos candidatos a stop de interesse do usuário.

O passo de clusterização, que identifica regiões de baixa velocidade, utiliza três parâmetros principais: *avg, speedLimit e minTime.* Inicialmente são encontradas as regiões de velocidade média abaixo do parâmetro *avg*, que representa a porcentagem média da velocidade. Esta etapa encontra todas as subtrajetórias com velocidade abaixo da velocidade média da trajetória inteira. Posteriormente essas regiões são expandidas, tentando-se agregar outros pontos de baixa velocidade. Dessa forma, vão sendo adicionados os pontos vizinhos que tenham velocidade inferior ao parâmetro velocidade máxima da trajetória (*SpeedLimit*) e que mantenham a velocidade média total da região sendo analisada, abaixo da velocidade média (*avg*). Terminada a expansão dessas regiões, o próximo passo é escolher as regiões com menor velocidade para se tornarem um stop potencial. Se este stop potencial tiver a duração maior que o parâmetro mínimo de tempo *minTime*, ele é considerado um cluster e um stop é gerado.

O segundo passo (atribuição de semântica geográfica) diz respeito ao contexto geográfico da trajetória. Nesse momento, os trechos de interesse provenientes do passo anterior (stops/clusters com baixa velocidade) são verificados se interceptam os candidatos a stop. Dessa forma, podem-se ter dois tipos de stops: *known stops* e *unknown stops*. Os primeiros estão relacionados a pontos de interesse do usuário que existem na base de dados de candidatos a stops. Os unknown stops, por outro lado, são pontos que foram capturados no primeiro passo como clusters (devido a sua baixa velocidade), mas que pela seleção de interesse do usuário ou falta de candidatos a stops na base de dados, não intercepta nenhum candidato a stop, sendo por isso chamado de *Unknown stop*.

A Figura 4 ilustra um exemplo do resultado do algoritmo onde há a descoberta de *unkown* e *kown* stops.

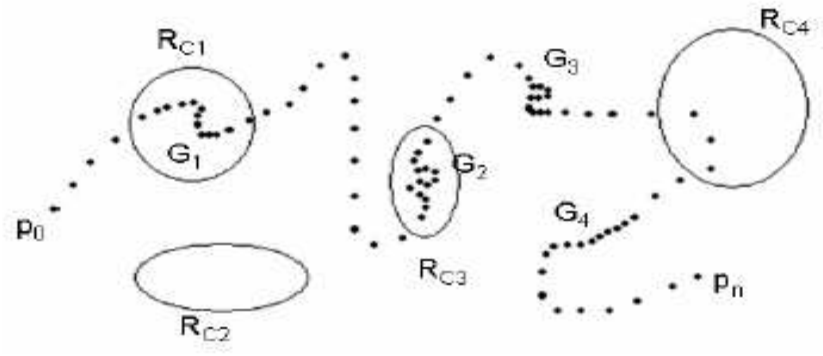


Figura 4– Exemplo de trajetória com 2 stops e 2 unknown stops (figura obtida em [Palma 2008]).

A trajetória da Figura 4 tem 4 stops possíveis, os clusters G_1, G_2, G_3 e G_4. Neste exemplo, o usuário informou 4 candidatos a stops, identificados pelas reticências R_{C1}, R_{C2}, R_{C3} e R_{C4}. O cluster G_1 intercepta o candidato a stop R_{C1} por um tempo superior a $_{C1.}$ Então, o primeiro stop da trajetória é R_{C1}. O mesmo ocorre com o cluster G_2, considerando R_{C3}, que é o segundo stop da trajetória. Os clusters, G_3 e G_4 não interceptam qualquer candidato a stop. Portanto, G_3 e G_4 são stops desconhecidos. Cada stop desconhecido recebe uma identificação. No caso de dois ou mais stops desconhecidos se cruzarem, eles vão receber a mesma identificação (por exemplo, 4_unknown).

3. MÉTODOS PARA REDUÇÃO DE DADOS

Como o método CB-SMOT é baseado na variação da velocidade das trajetórias, ele é bastante interessante para aplicações de trânsito e por isso foi escolhido como o principal algoritmo do objeto de estudo deste trabalho. Trajetórias geradas por GPS em carros normalmente produzem um grande volume de dados. Considerando a demora de execução do algoritmo, verificou-se a necessidade de reduzir desse tempo de processamento. Assim sugiram duas opções: otimizar o algoritmo ou reduzir o volume de dados. Ao reduzir o volume de dados, outros algoritmos automaticamente serão beneficiados. Optou-se então por reduzir a quantidade de dados utilizando algumas técnicas de amostragens estatísticas como principais formas de otimização de tempo de processamento.

Uma das preocupações deste trabalho é que as técnicas de redução aplicadas garantam que a trajetória resultante da remoção e limpeza de dados represente semanticamente a trajetória original. Para isso, aplicaram-se algumas técnicas de amostragem usando alguns conceitos estatísticos de população e amostra.

Barbeta [Barbeta 2001] definiu população como sendo um conjunto de elementos passíveis de serem mensurados, com respeito às variáveis que se pretende levantar, podendo ser formada por pessoas, famílias, estabelecimentos industriais, ou qualquer outro tipo de elementos, dependendo basicamente dos objetivos da pesquisa. Considerando que o objeto de pesquisa deste trabalho é principalmente as trajetórias e seus respectivos stops, então população pode ser considerada como uma trajetória.

Para Barbeta [Barbeta 2001], a população alvo é o conjunto de elementos que se quer abranger em um estudo, ou seja, os elementos para os quais se deseja que as conclusões oriundas da pesquisa sejam validadas.

O termo amostragem [Lavado 2001] refere-se ao processo pelo qual se obtém uma amostra e deve ser realizada com técnicas adequadas para garantir a representatividade da população em estudo. Assim, essas técnicas têm que garantir que as trajetórias limpas representem a trajetória original. Ele ainda afirma que, sempre que possível, cada elemento da população deve ter igual chance de participar da amostra, evitando assim, o chamado viés de seleção. Isto significa que cada ponto da trajetória original tem que ter a mesma probabilidade de participar da trajetória limpa sem haver qualquer tipo de tendência ou influência.

[Barbeta 2001] afirma que técnicas de amostragem extraem do todo (população) uma parte (amostra) com o propósito de avaliar (inferir) algumas características da população, ou seja, a partir de técnicas de amostragem retira-se amostras que representem as características da população. Justificativas para o uso de amostragens:

1) *Economia.* Torna-se mais econômico o levantamento de somente uma parte da população.
2) *Tempo*. Para uma população consideravelmente grande, não há tempo viável de avaliar toda a população.
3) *Operacionalidade.* É mais fácil realizar operações de pequena escala.

Todas as três razões se justificam neste trabalho, uma vez que, ao se selecionar menos dados, o algoritmo executa mais rapidamente e como o volume de dados (população) também é grande, se torna inviável a execução do algoritmo devido ao tempo de processamento.

As formas de limpeza das trajetórias adotadas neste trabalho são compostas por dois métodos de amostragem e um de eliminação de ruídos, os quais são discutidos a seguir.

3.1. Eliminação de ruído

Ruído é considerado algum valor de velocidade ou tempo que esteja variando acima do parâmetro informado pelo usuário.

O usuário informa a velocidade ou o tempo máximo entre dois pontos da trajetória a ser limpa. O sistema calcula a velocidade e o tempo de todos os pontos da trajetória original. O algoritmo de limpeza verifica se a velocidade ou o tempo de cada ponto da trajetória original é maior que o parâmetro informado. Caso seja, então o ponto avaliado é removido da trajetória original. Caso contrário, o ponto fará parte da trajetória limpa.

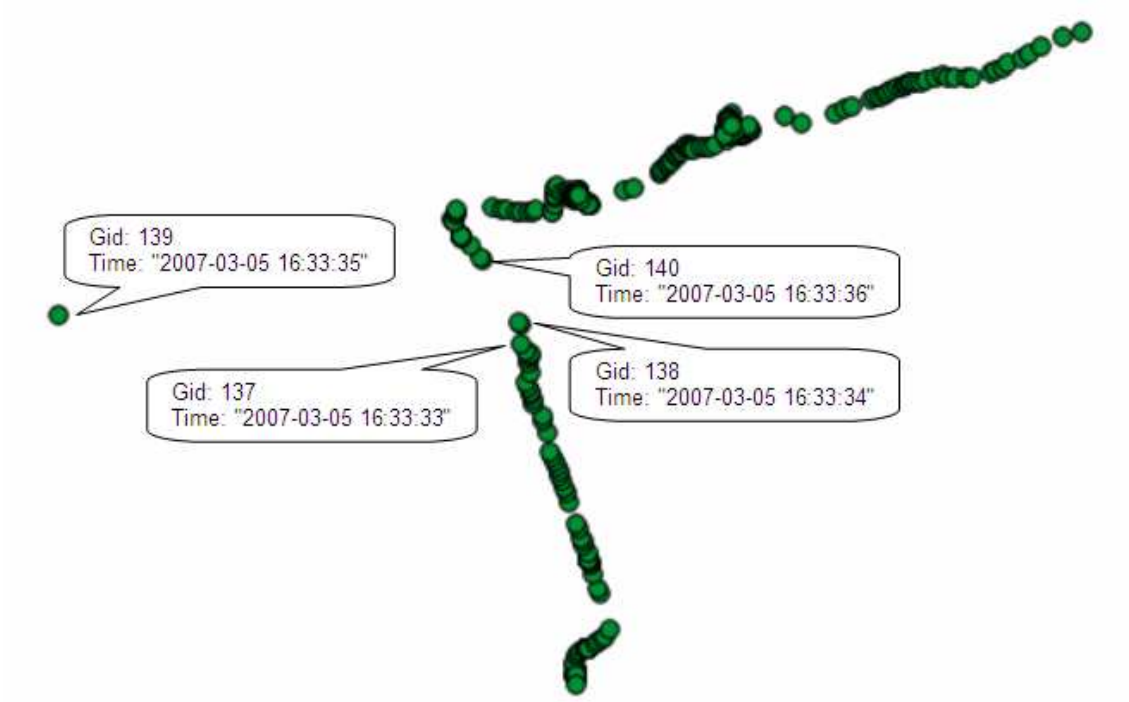


Figura 5 – Exemplo de ponto gerado errado

A Figura 5 mostra um exemplo de um ponto gerado com erro em sua coluna geometria. Observa-se, pelo atributo *time*, que os pontos estão variando a cada segundo e pelo atributo *gid*, que estão variando seqüencialmente. Já se observarmos a distância percorrida pelos pontos, notaremos que o ponto de *gid* 139 percorreu uma distância consideravelmente maior que a distância dos pontos restantes.

Sabe-se que a velocidade de um corpo é calculada pela divisão da distância percorrida pelo período de tempo do percurso. Com isso, concluímos que a velocidade do ponto 139 é muito maior que as velocidades dos demais pontos, sendo impossível o objeto percorrer aquele trajeto naquela velocidade. Assim sendo, este objeto é excluído no processo de limpeza de trajetórias sem perda de informação.

Esta técnica de redução de dados muito se assemelha à forma como Pelekis [Pelekis 2008] utiliza os parâmetros gapTime e VMax na seção 3.1, que determinam se o ponto que está sendo avaliado do conjunto de pontos brutos vai fazer parte ou não da trajetória comparada. O primeiro parâmetro determina o máximo permitido de intervalo de tempo entre dois pontos, que pode ser comparado ao parâmetro tempo máximo proposto. Já o segundo determina a velocidade máxima permitida de um ponto na trajetória, que tem o mesmo sentido do parâmetro velocidade máxima proposto nesta técnica. A diferença entre a técnica proposta e a técnica de Pelekis é que, na técnica de

Pelekis, os pontos que são considerados como ruídos não são removidos inicialmente, uma vez que, estes pontos podem ser interessantes para outras trajetórias futuras. No entanto, as trajetórias finais reconstruídas não possuem os pontos ruidosos.

3.2 Amostragem Aleatória Simples

A amostra aleatória simples tem a seguinte propriedade [Barbeta 2001]: *qualquer subconjunto da população, com o mesmo número de elementos, tem a mesma probabilidade de fazer parte da amostra*. Em suma, tem-se que cada ponto da trajetória tem a mesma probabilidade de pertencer à amostra (trajetória limpa).

O usuário informa a porcentagem da trajetória original que vai fazer parte da trajetória limpa, ou seja, ele informa o tamanho da amostra, que não deixa de ser, o tamanho da trajetória limpa. Por exemplo, se para o usuário só interessar 50% dos dados da trajetória original, então ele informa ao sistema esta quantia e este sorteia 50% dos dados da trajetória original para a trajetória limpa e salva em uma nova tabela no banco de dados. Assim, temos uma nova tabela com menos pontos nas trajetórias.

3.3. Amostragem sistemática

Em [Unama 2010], os membros da população (pontos da trajetória) que participam da amostra são determinados a partir de intervalos fixos, e pode-se somente selecionar aleatoriamente o primeiro atributo (ponto). Por exemplo, uma amostra de pacientes internados em uma clínica. Sorteia-se um número de 1 a 5. Se o número sorteado for 2, incluem-se na amostra o paciente 2, o 7, o 12 e assim por diante variando de cinco em cinco.

Nesta técnica o usuário informa quantos pontos quer retirar entre dois pontos da trajetória a ser limpa. O primeiro ponto a ser selecionado é sorteado aleatoriamente entre 1 e 5. Os posteriores serão selecionados conforme o parâmetro determinado pelo usuário, ou seja, se o quinto(5°) ponto da trajetória original foi sorteado aleatoriamente como o primeiro ponto da trajetória limpa e o usuário tenha informado 4 como valor do parâmetro, então a trajetória limpa será constituída pelos pontos (5, 9, 13, 17, ...) da trajetória original.

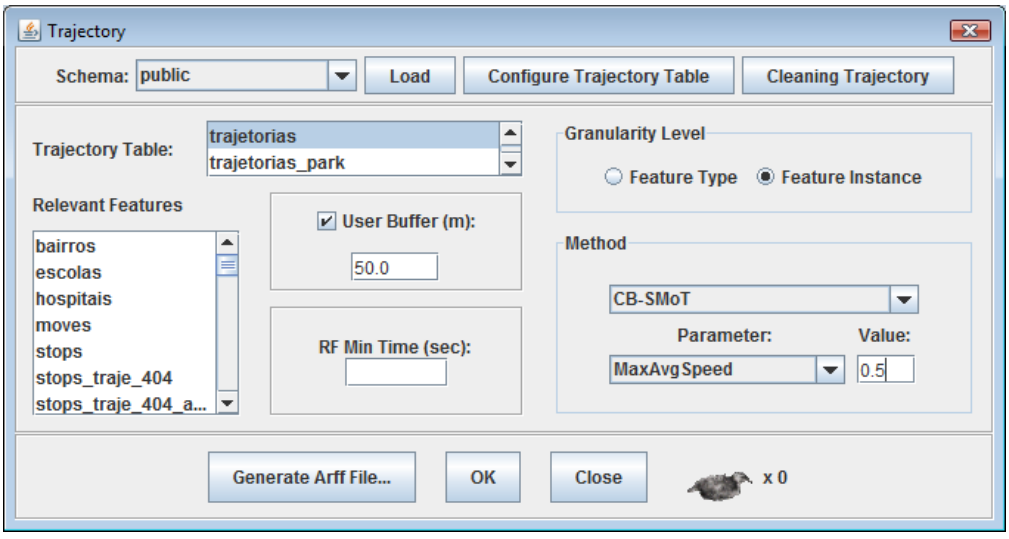
3.4. Implementação

Como os algoritmos de extração de stops e moves estão implementados na ferramenta Weka [Frank 2005], as técnicas de redução de dados propostas neste trabalho também foram implementadas nesta ferramenta.. O Weka é uma das ferramentas de mineração de dados mais utilizada em meios acadêmicos, desenvolvida em Java, de código aberto e para dados não-espaciais. Este kit possui uma coleção de algoritmos de aprendizado de máquina para tarefas de mineração de dados. Ele tem um módulo de processamento de dados não-espaciais chamado "Explorer". Os dados podem ser obtidos a partir de um banco de dados, um site ou um arquivo ARFF (arquivo de entrada de texto no formato requerido pelo Weka).

O Weka já foi estendido pelo nosso grupo de pesquisa para suportar a mineração de dados geográficos [Bogorny 2006] e recentemente foi estendido para suportar o pré-processamento e a mineração de trajetórias [Álvares 2010].

A extensão para dados de trajetórias é chamada módulo STPM, que é totalmente integrado no Weka, a fim de acessar automaticamente o banco de dados geográfico e adicionar semântica aos dados de trajetórias. Este Weka é interoperável com todos os bancos de dados, pois é desenvolvido sobre as especificações do Open GIS Consortium (OGC) [OGC 2008].

Considerando que todas as variações do Weka citadas anteriormente são aplicativos de código aberto, resolveu-se criar um sub-módulo de limpeza de trajetórias dentro do modulo de STPM. Assim, o usuário faz a limpeza dos dados, o pré-processamento e a mineração dentro de uma única ferramenta.

a) 

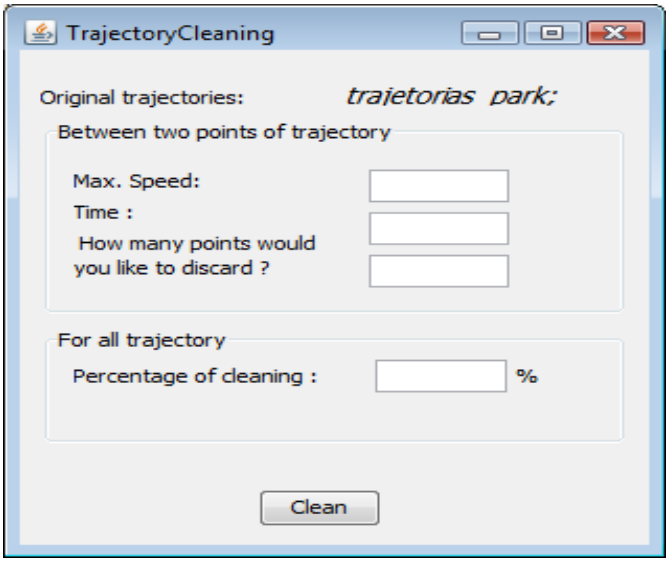
b) 

Figura 6 – a) Tela de seleção de trajetórias. b) Tela de limpeza de trajetórias.

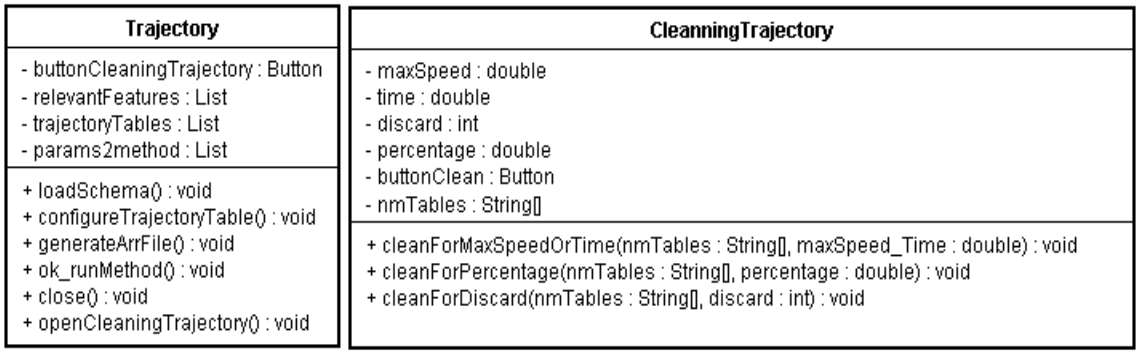


Figura 7 – Diagrama das classes das telas alteradas ou inseridas. a) Atributos e métodos adicionados à tela Trajectory. b) Tela "Cleanning Trajectory" adicionada ao modulo STPM.

O item a da Figura 6a mostra a tela original de seleção de trajetórias do módulo STPM do Weka com o botão "Cleaning trajetory". Este botão foi adicionado para abrir a tela de limpeza. Ele chama o método "openCleaningTrajectory()", Figura 7a, que verifica se foi selecionada pelo menos uma tabela de trajetórias no item *Trajectory Table* da Figura 6a. Caso contrário, uma mensagem é informada ao usuário, que necessita informar ao menos uma tabela a ser limpa.

Já na tela de limpeza de trajetórias, na Figura 6b, observa-se que a tela está dividida em dois painéis, que contêm os parâmetros de entrada de cada método de limpeza, visto na Figura 7b, sendo que apenas um parâmetro pode ser selecionado para as técnicas de limpeza. Cada painel representa uma forma de aplicação das técnicas de limpeza descrita no Capitulo 3.

No primeiro painel, o usuário pode informar a velocidade máxima, o tempo, ou quantos pontos ele deseja retirar entre dois pontos da mesma trajetória. Caso seja

informada a velocidade máxima ou o tempo, será executado método "cleanForMaxSpeedOrTime()" da Figura 7b, que representa a técnica de redução de dados citada na seção 3.1 deste trabalho. Já se o parâmetro "*How many points would you like to discard?"* for informado, então será executado o método "cleanForDiscard()" da Figura7b que representa a técnica de amostragem sistemática descrita na seção 3.3.

No segundo painel, se for informado o parâmetro "*Percentage of cleaning*", então será executado o método "cleanForPercentage()" da Figura 7b que representa a técnica de amostragem aleatória simples descrita no item 3.2 deste capítulo.

Na Figura 6b, além dos dois painéis, existe o botão "clean". A partir do momento em que esse botão é pressionado, verifica-se se o usuário informou os campos corretamente, então o método de limpeza é executado para cada tabela de trajetórias original selecionada. O método de limpeza cria uma nova tabela no banco de dados com os mesmos atributos da tabela de trajetórias original. Nesse instante, os registros de cada trajetória da tabela original são selecionados conforme a técnica de limpeza informada nos painéis da Figura 6b e inseridos na nova tabela criada no início do método. Assim, o método de limpeza se repete até que todas as tabelas de trajetórias selecionadas sejam limpas pelas técnicas de limpeza.

4. Experimentos e Análise dos resultados

A seguir são mostrados os experimentos e análises dos resultados da aplicação das técnicas de redução de dados sobre dois conjuntos de trajetórias: carros e pedestres. No entanto, houve dificuldade em achar o limite do número ou do percentual de pontos ideal em que se pode remover da trajetória original sem alteração nos resultados. Devido a esse valor depender exclusivamente da redundância de dados de cada conjunto, foram fixados os mesmos parâmetros de redução de dados para ambos os conjuntos, com o objetivo de avaliar se a aplicação de técnicas de redução compromete a semântica dos stops originais. Para técnica de redução por amostragem sistemática da seção 3.3 foram usados os parâmetros 1, 2, 3 e 4. Já para técnica de amostragem aleatória simples foram usados 50%, 33%, 25% e 20% como parâmetros.

4.1. Conjunto de trajetórias de carros

O primeiro conjunto de dados utilizado para avaliar as técnicas foram dados gerados por aparelhos de GPS em automóveis que transitaram pela cidade do Rio de Janeiro entre o período 24 de fevereiro de 2002 a 16 de agosto de 2007. Esses dados constituem uma tabela no banco de dados com 1.000.000 de registros e contendo 325 trajetórias diferentes e em média 3.076 pontos por trajetória.

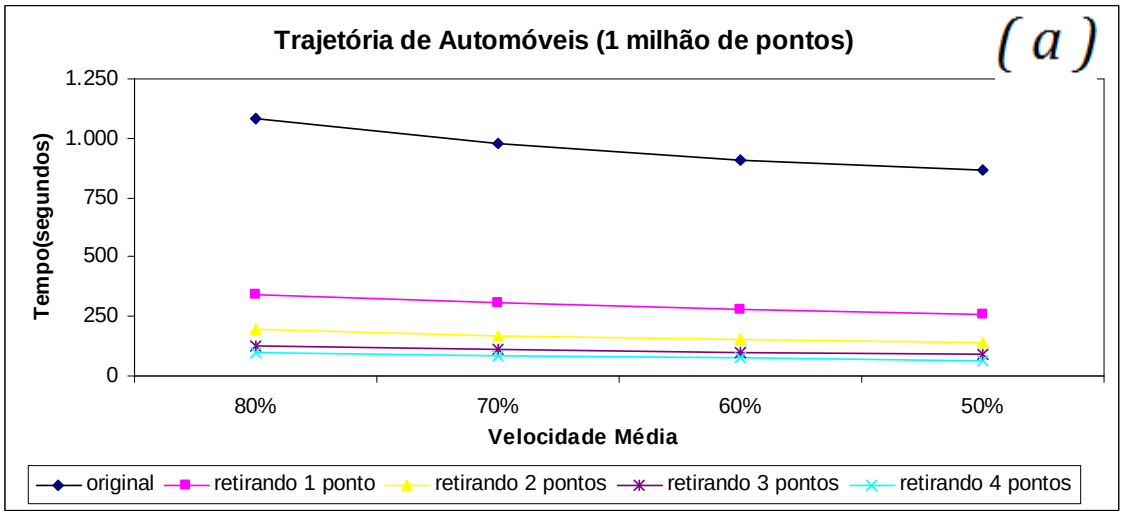
4.1.1. Análise do tempo de processamento

Para melhor avaliar o tempo de processamento do algoritmo CB-SMOT, foram fechados todos os programas adicionais do sistema operacional, salvo os realmente necessários para a execução do aplicativo Weka. Mesmo assim, os resultados podem ser divergentes, se executados em máquinas e sistemas operacionais diferentes. Em todos os casos, o algoritmo foi executado em um sistema operacional Windows Vista Home Basic, versão 1.6.0.20 do Java Development Kit(JDK), versão 8.4 do banco de dados Postgres, versão 1.4 do Postgis (complemento para consultas geográficas do Postgres) para uma máquina de 3 mega de memória RAM e processador Intel Core 2 Duo.

O Gráfico 1 representa o tempo de processamento do algoritmo CB-SMOT pela variação do parâmetro Velocidade Média (Average Speed) 0,5(50%) a 0,8(80%) da

velocidade média de toda a trajetória. Os parâmetros fixos utilizados foram o Tempo Mínimo do stop (MinTime) com 600 segundos (10 minutos) e o Limite de Velocidade (Speed Limit) de 1,1. O Gráfico 1 utiliza uma tabela de trajetórias de automóveis que percorrem as vias do cidade do Rio de Janeiro. Para a tabela do gráfico 1a foi aplicada a técnica de amostragem sistemática, com os filtros 1, 2, 3 e 4 pontos, gerando assim quatro novas tabelas com respectivamente 500 mil, 330 mil, 250 mil e 200 mil pontos. Já para a tabela do gráfico 1b, foi aplicada a técnica de amostragem aleatória simples, citada no item 3.2, com os filtros 50%, 33%, 25% e 20%, gerando assim quatro novas tabelas com respectivamente 499.695, 333.174, 249.918 e 182.503 pontos.

A tabela original, que possui um milhão de pontos, é representada pela primeira e maior linha do gráfico. As demais linhas representam a tabela original com os dados reduzidos, podendo ser melhor conferidas na legenda do gráfico. No gráfico 1 pode ser observado que a exclusão de um único ponto, visto no gráfico 1a, ou a utilização de apenas 50% da trajetória original, visto no gráfico 1b, reduziu mais de 50% do tempo de processamento do algoritmo, independente da técnica de redução de dados.



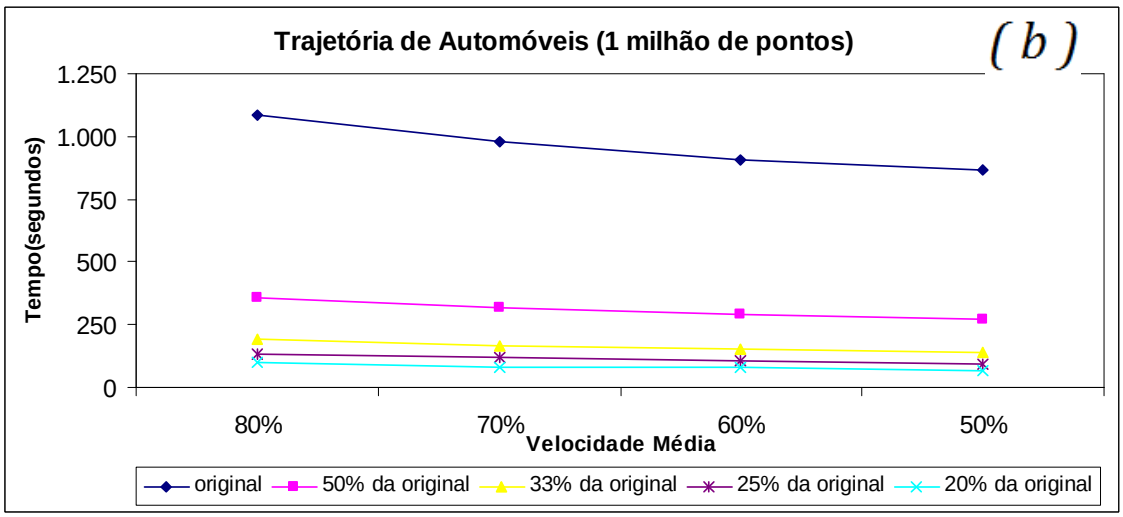


Gráfico 1 - Redução do tempo de processamento das trajetórias de carros.

A Tabela 2 esclarece o percentual de otimização que ocorre com a execução da limpeza dos dados em relação à tabela original para a execução do algoritmo CB_SMOT com 50% da velocidade média da trajetória. Note que, independente da técnica de redução de dados, a retirada de apenas um ponto ou a utilização de apenas 50% da trajetória original resulta em uma redução de 70% do tempo de processamento do algoritmo. Quanto maior a limpeza dos dados, menor é o tempo de processamento.

Tempo de processamento	Trajetórias Limpas	Trajetória Original	Porcentagem de Otimização
retirando 1 ponto	259 seg	864 seg	70%
retirando 2 pontos	139 seg	864 seg	83%
retirando 3 pontos	89 seg	864 seg	89%
retirando 4 pontos	54 seg	864 seg	93%
50% da original	273 seg	864 seg	68%
33% da original	140 seg	864 seg	83,7%
25% da original	94 seg	864 seg	89%
20% da original	66 seg	864 seg	92%

Tabela 2 - Porcentagem de otimização do tempo de processamento do algoritmo CB-SMOT .

4.1.2. Análise do total de stops gerados

Nesta seção avaliaremos a qualidade do resultado do algoritmo após a limpeza dos dados.

Observando o Gráfico 2, é perceptível que, independente do tipo de limpeza utilizada, a quantidade de stops varia conforme aumenta a limpeza da trajetória original. Entretanto, pode-se observar que a variação no número de stops gerados após a limpeza

é pequena para este conjunto de dados. Como mostra o Gráfico 2a, para uma velocidade média de 50%, a retirada de apenas um ponto diminuiu um stop, já a retirada de 2 ou 3 pontos aumentam 4 stops na trajetória limpa.

Nota-se, observando o Gráfico 2, que independente do tipo de limpeza utilizada existem três grupos de resultados diferentes: quando não há alteração na quantidade de stops entre as trajetórias originais e limpas, quando a quantidade de stops das trajetórias limpas é maior que o número de stops da trajetória original e também quando ocorrer o contrário.

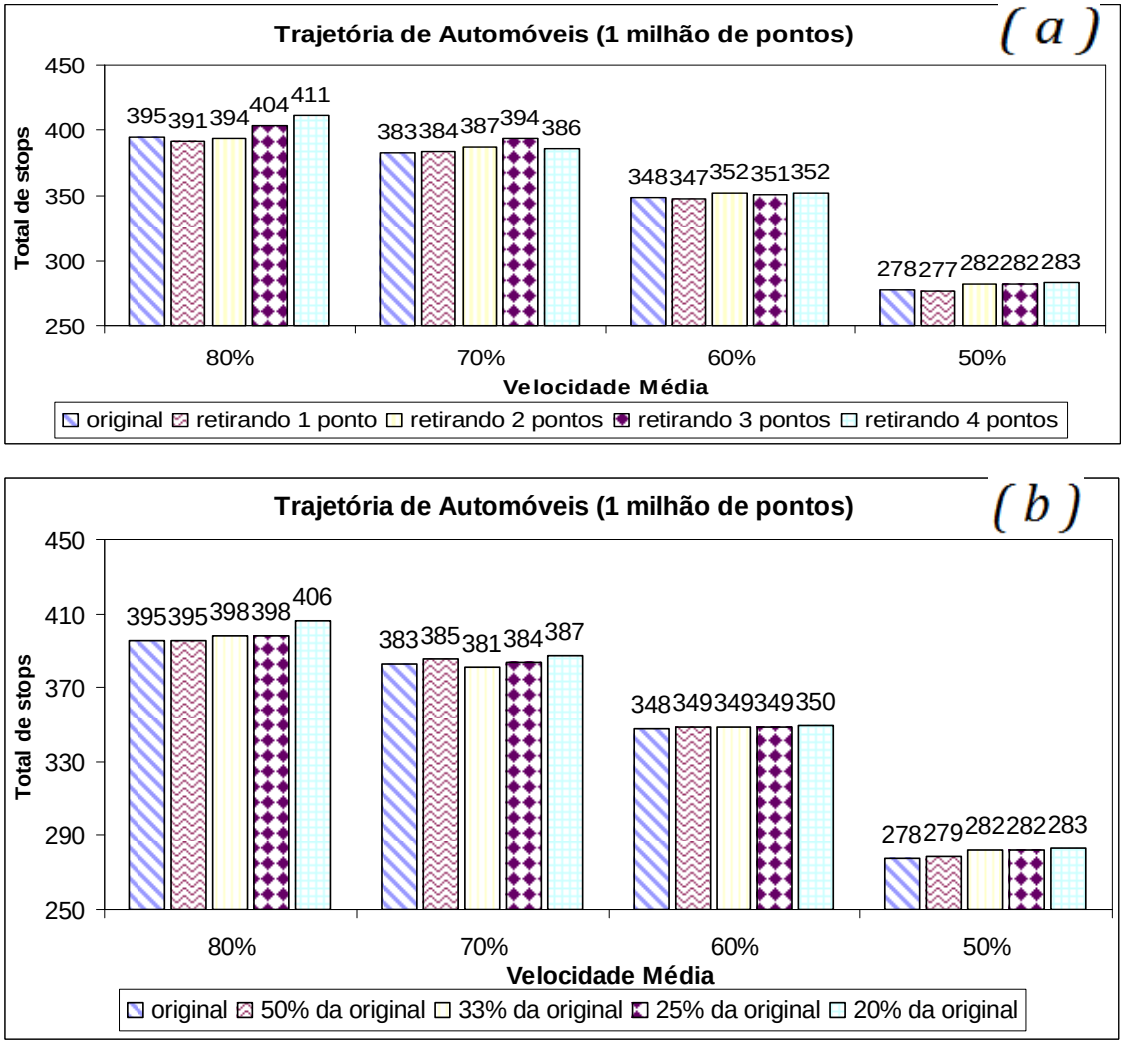


Gráfico 2 - Quantidade de stops por tabela.

O resultado ideal seria não haver alteração entre os stops obtidos após a execução do algoritmo sobre as tabelas originais e limpas. No entanto, as diferenças, embora pequenas, acontecem devido à utilização dos métodos de limpeza. Um exemplo de limpeza e execução ideal do algoritmo pode ser observado na Figura 8, que mostra uma trajetória com os stops gerados sobre a trajetória original e os stops gerados após a

retirada de um até quatro pontos. Note que os stops não são exatamente iguais, sempre há uma pequena diferença, aceitável, devido à retirada dos pontos. O mais importante é que os stops estão na mesma região, representando a mesma área geográfica e tem a mesma semântica.

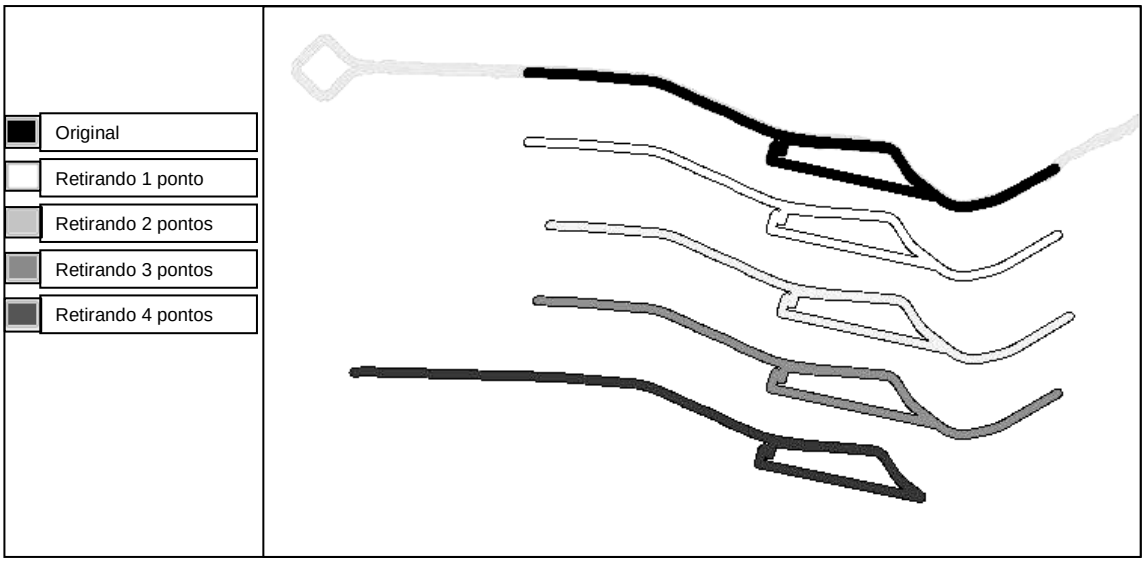


Figura 8 – Exemplo de limpeza com remoção de até 4 pontos.

Nos casos em que a trajetória limpa gerou menos stops que a trajetória original, estes stops sempre estão próximos ou até mesmo sobrepostos com os da trajetória original, podendo também ser de tamanho maior, considerando assim uma junção de dois stops originais. A Figura 9 ilustra um exemplo, onde os dois stops inferiores da trajetória original foram aglutinados em apenas um stop nas trajetórias limpas. Note que os stops que se interceptam na trajetória original possuem o mesmo nome (*9_unknown*). Assim, observa-se que ambos os stops caracterizam uma mesma região lenta. Em seguida, observe que os stops das trajetórias limpas também possuem o mesmo nome (*9_unknown*). Com isso, entende-se que não houve perda de informação, a semântica dos stops é preservada, embora sua geometria tenha sido alterada.

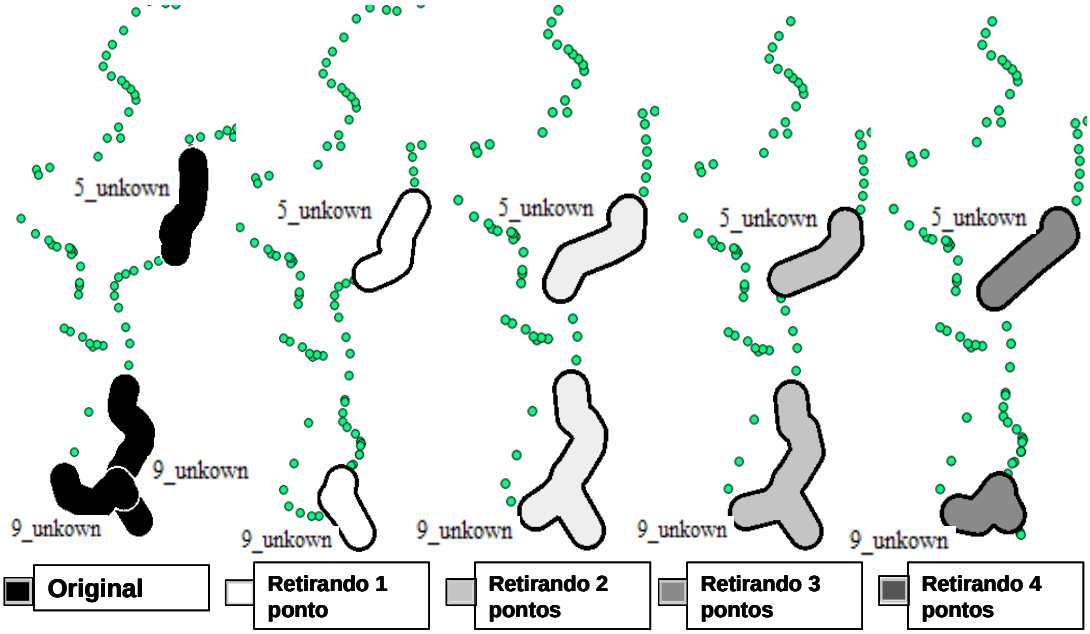


Figura 9 – Exemplo de limpeza e execução do algoritmo CB-SMOT onde os stops originais são em maior número que os stops das trajetórias limpas

Nos casos em que são gerados mais stops, há duas explicações plausíveis: No primeiro caso, considera-se que alguns pontos importantes do stop sejam excluídos na limpeza, e conseqüentemente quando os stops são calculados sobre as trajetórias limpas, o stop original se divide em dois ou três novos stops. A Figura 10 ilustra um stop original de tamanho grande que foi dividido a partir da retirada de dois pontos ou mais nas trajetórias limpas. Observe que, assim como na Figura 9, os nomes se mantiveram independente da aplicação das técnicas de limpeza. Embora tenham sido gerados mais stops após a limpeza dos dados, em todos os casos os stops permanecem como *0_unkown*. Com isso, entende-se que a semântica dos stops se mantém independente da retirada de um ou mais pontos.

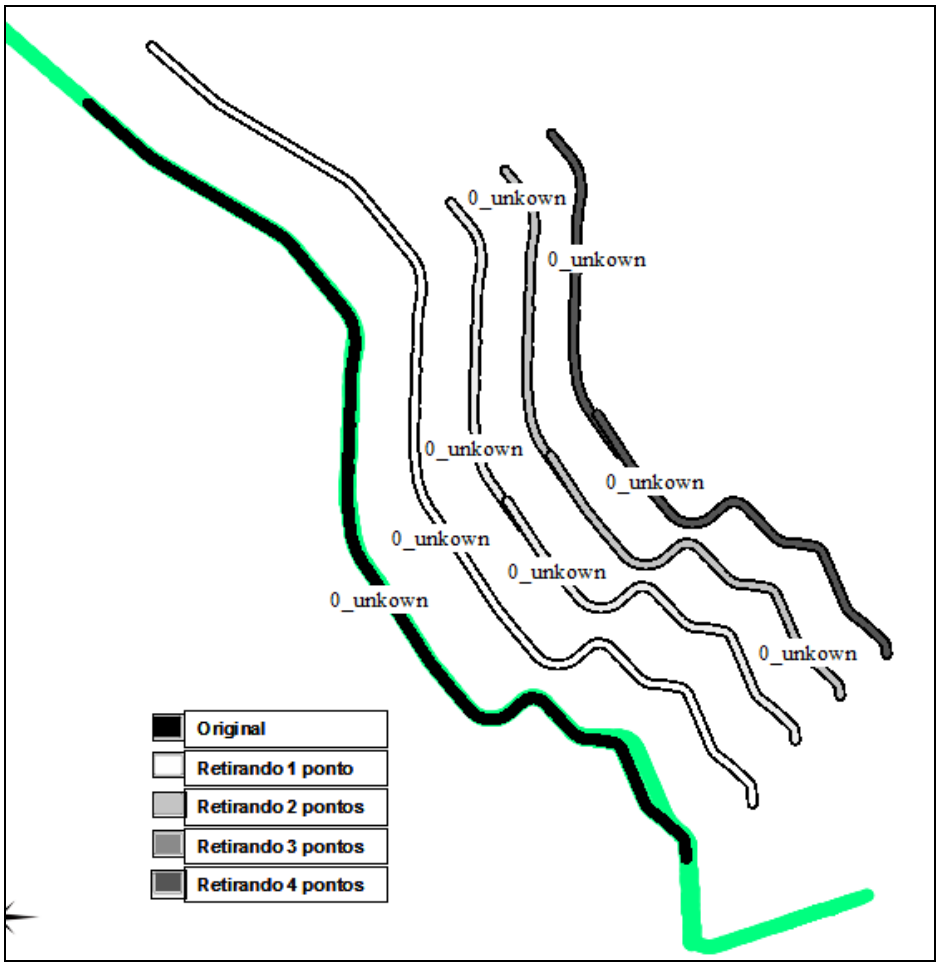


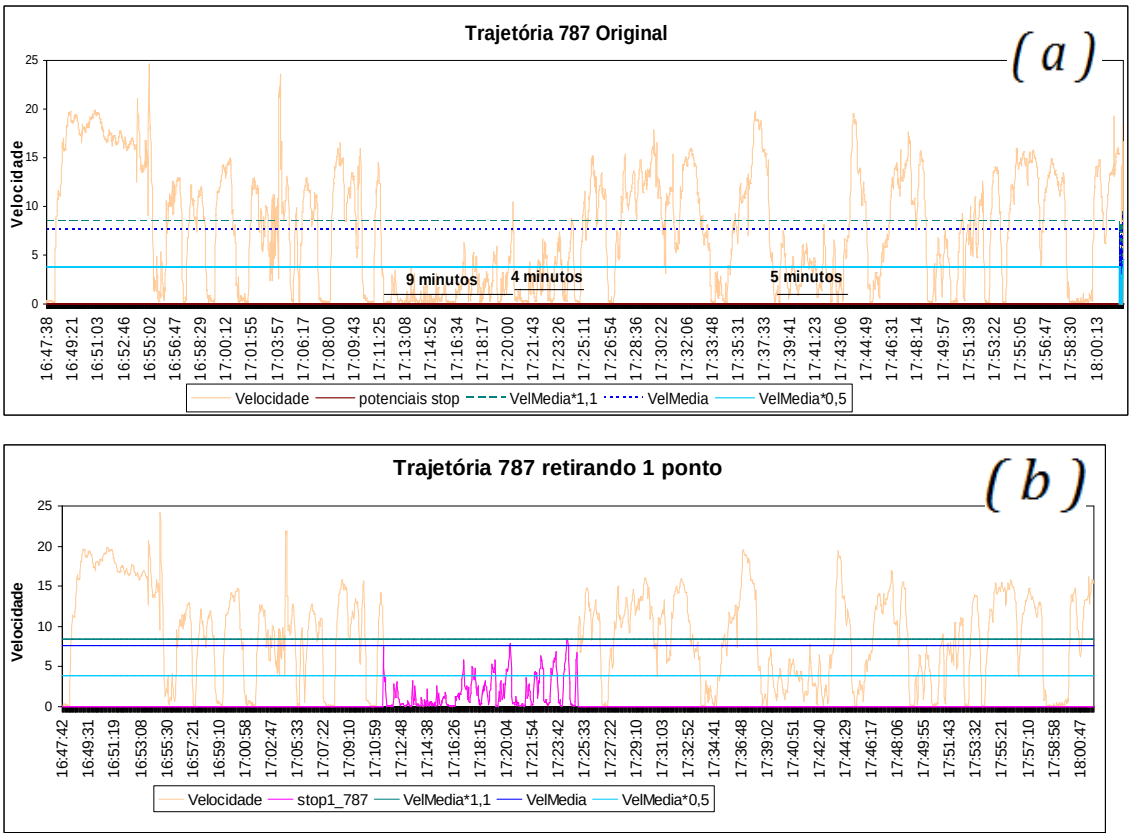
Figura 10 – Exemplo de limpeza e execução do algoritmo CB-SMOT na qual os stops originais foram divididos em dois os stops nas trajetórias limpas.

No segundo caso, a quantidade de stops das trajetórias limpas também é em quantidade maior, ocorrendo devido a potenciais stops (clusters) com velocidade baixa estarem na mesma região, mas sem tempo mínimo de duração para ser considerado um stop na trajetória original. Quando ocorre a limpeza, os pontos que os separavam são removidos e, dessa forma, ambos serão considerados como apenas um stop potencial e se este tiver o tempo mínimo, então será um novo stop na trajetória limpa que não existia na trajetória original. Este caso pode ser observado nos gráficos 3a, 3b, 3c e 3d, onde a linha de corte é a velocidade média.

O Gráfico 3 foi gerado usando as velocidades de cada ponto da trajetória e os stops gerados aplicando os seguintes parâmetros: 0,5 de velocidade média, 600 segundos de tempo mínimo e 1,1 de limite de velocidade. Os gráficos 3a, 3b, 3c, 3d e 3e representam a trajetória 787 e suas trajetórias limpas da tabela de trajetórias de automóveis do Rio de Janeiro. Para as trajetórias limpas foi executada a técnica de amostragem sistemática

com a remoção de um até quatro pontos. A trajetória original 787 possui o total de 4.274 pontos. As trajetórias limpas 787 retirando 1 ponto possui 2.135 pontos, retirando 2 pontos possui 1.423 pontos, retirando 3 pontos possui 1.068 pontos e a retirando 4 pontos 855 pontos. Note que a diferença na quantidade de pontos quase não é perceptível, apesar da diferença na quantidade de dados chegar a 80% entre os gráficos "a" e "e".

A trajetória original (Gráfico 3a) não gerou nenhum stop, sendo grifado com uma linha apenas o stop potencial com seu respectivo tempo de duração. Note que entre os potenciais stops de 9 minutos e 4 minutos existem apenas alguns pontos e estes ultrapassam o parâmetro de limite de velocidade, fazendo com que fiquem separados e acarretando a não geração do stop, pois ambos não possuem o tempo mínimo de 600 segundos (10 minutos) definido para a geração dos stops. Agora observe os gráficos restantes do Gráfico 3 e note que onde eram os potenciais stops de 9 e 4 minutos foi gerado um stop para todas as trajetórias retirando de um até a de quatro pontos, juntando os pontos de baixa velocidade.



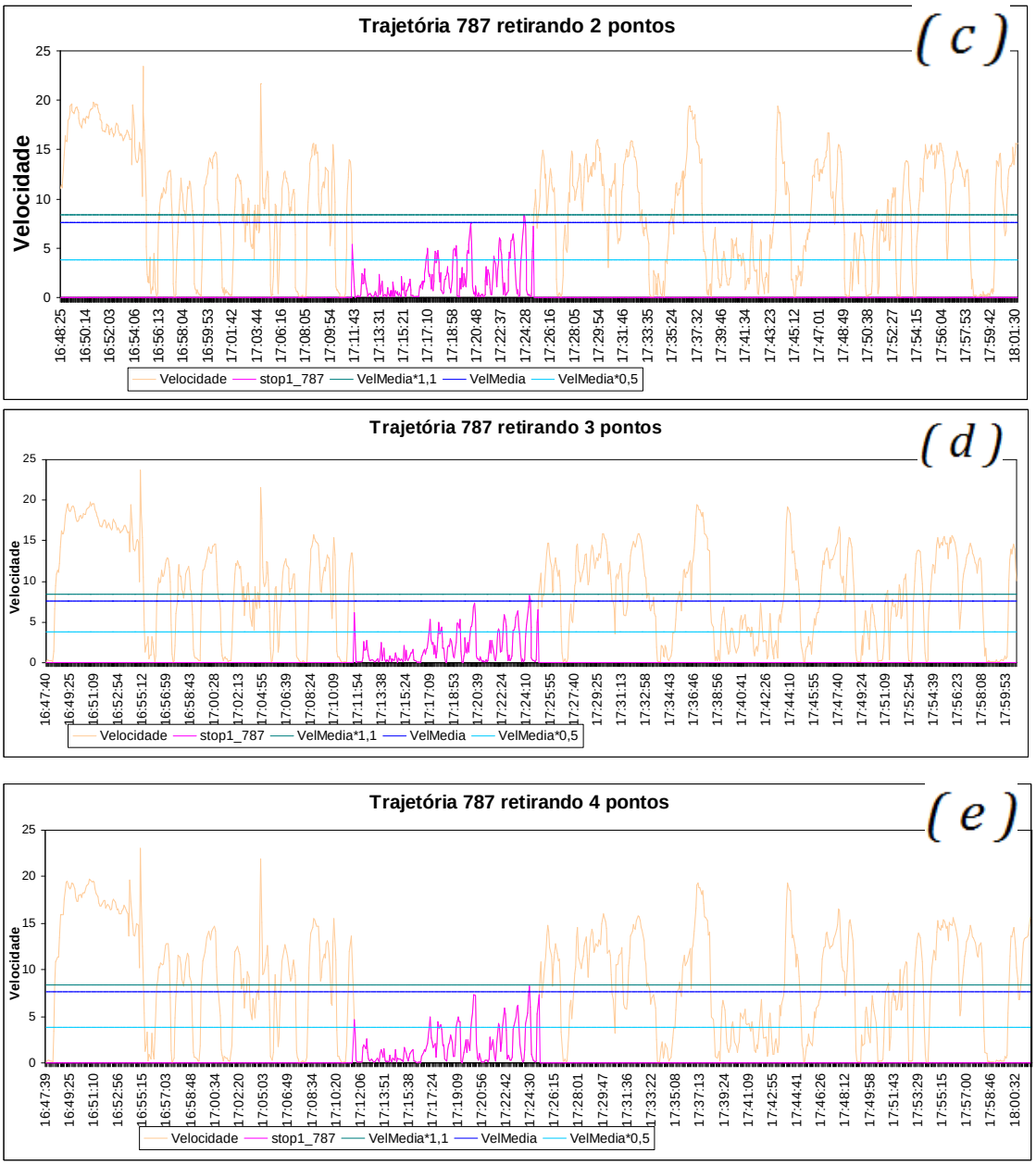


Gráfico 3 - Gráficos de velocidade pelo tempo de cada ponto da trajetória 787, com stops ou candidatos a stop das trajetórias original e limpas.

A próxima seção apresenta os experimentos realizados sobre outro conjunto de dados reais.

4.2. Conjunto de trajetórias de pedestres

O segundo conjunto de dados utilizado para avaliar os experimentos foram dados de trajetórias de pedestres da cidade de Amsterdam do período 18 de maio de 2006 a 19 de agosto de 2006. O conjunto de dados tem 151.576 registros, contendo 372 trajetórias diferentes e em média 408 pontos por trajetória. Em comparação ao conjunto de dados I,

nota-se que este conjunto é menor e a média da quantidade de pontos das suas trajetórias é menor, podendo assim gerar resultados diferentes.

Além disso, o intervalo de tempo entre os pontos do conjunto de dados anterior era de 1 segundo e as trajetórias eram de carros. Neste segundo experimento o intervalo de tempo em que os pontos foram gerados é em média de 17 segundos. Assim, já se sabe que as técnicas de limpeza de dados são mais apropriadas para dados muito densos, como o primeiro conjunto.

4.2.1. Análise do tempo de processamento

Os mesmos procedimentos e recursos físicos do conjunto de trajetórias de carros foram utilizados para a execução e análise do tempo de processamento deste conjunto de dados.

O Gráfico 4 representa o tempo de processamento do algoritmo CB-SMOT pela variação do parâmetro Velocidade Média (Average Speed) do algoritmo, variando de 0,5(50%) a 0,8(80%) da velocidade média de toda a trajetória. Os parâmetros fixos utilizados foram o Tempo Mínimo do stop (MinTime) com 600 segundos (10 minutos) e o Limite de Velocidade(Speed Limit) de 1,1. O Gráfico 4 utiliza uma tabela de trajetórias dos pedestres que percorrem um parque na cidade de Amsterdam. Para a tabela do Gráfico 4a foi aplicada a técnica de amostragem sistemática, citada no item 3.3, com os filtros 1, 2, 3 e 4 pontos, gerando assim quatro novas tabelas com respectivamente 75.448 pontos, 50.348 pontos, 37.796 pontos e 30.281 pontos. Já para a tabela do Gráfico 4b foi aplicada a técnica de amostragem aleatória simples, citada no item 3.2, com os filtros 50%, 33%, 25% e 20%, gerando assim quatro novas tabelas com respectivamente 75.688, 49.839, 37.744, e 30.168 pontos.

A tabela de trajetórias original, que possui 151 mil pontos, é representada pela primeira linha do gráfico. As demais tabelas (limpas) são representadas em seguida, podendo ser conferido melhor na legenda do gráfico.

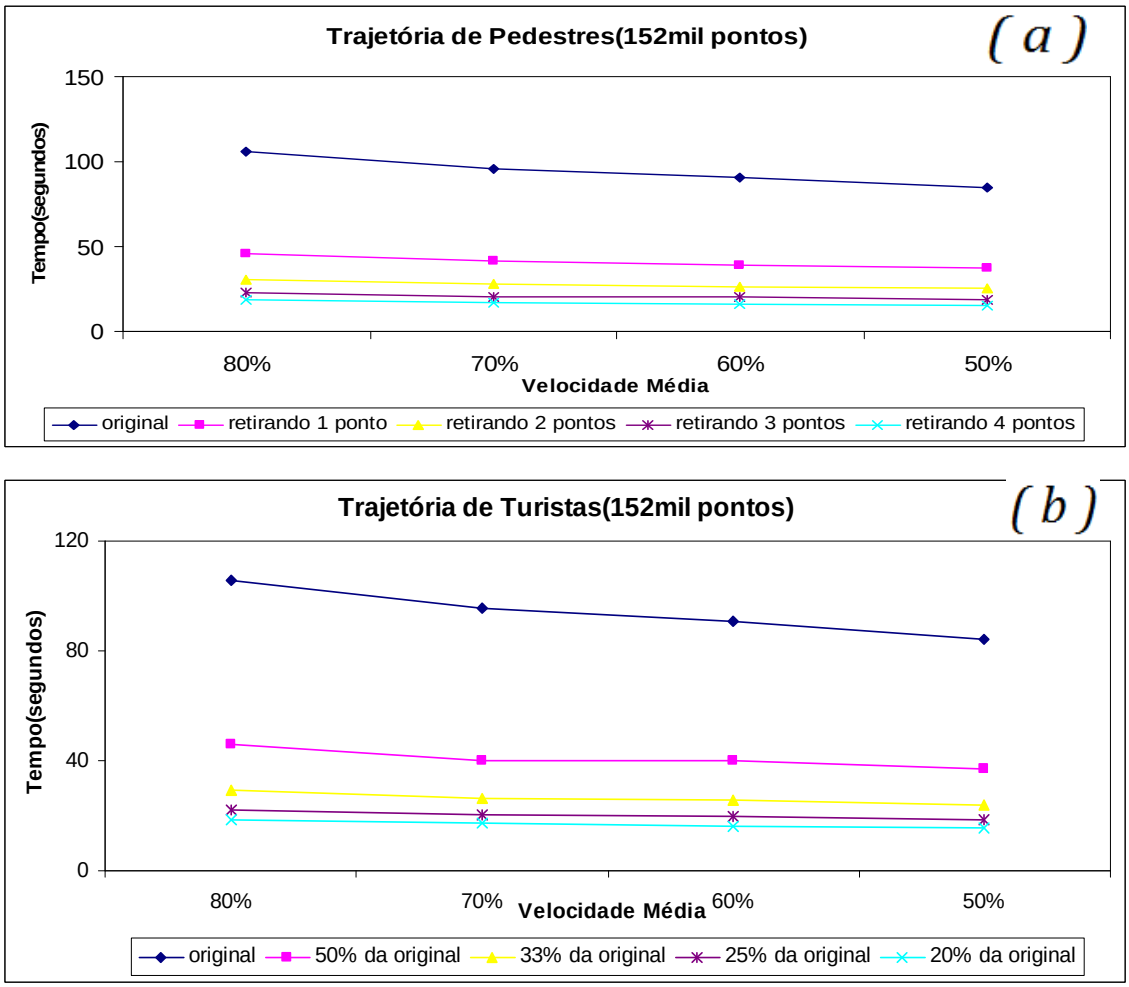


Gráfico 4 - Redução do tempo de processamento para trajetórias de pedestres.

Pode-se observar que os ganhos do tempo de processamento são similares ao primeiro conjunto de dados, havendo uma redução de mais de 50% no tempo de processamento com a eliminação de apenas um ponto ou com a utilização de apenas 50% da trajetória original.

A tabela 4 esclarece o percentual de otimização que ocorre com a execução da limpeza dos dados em relação à tabela original para a execução do algoritmo CB_SMOT com 50% da velocidade média da trajetória. Note que a retirada de apenas um ponto ou a utilização de apenas 50% da trajetória original surte em uma redução de 66% do tempo de processamento do algoritmo e quanto maior a limpeza dos dados, menor é o tempo de processamento.

Tempo de processamento	Trajetórias Limpas	Trajetória Original	Porcentagem de Otimização
retirando 1 ponto	37 seg	84 seg	66%
retirando 2 pontos	25 seg	84 seg	70%

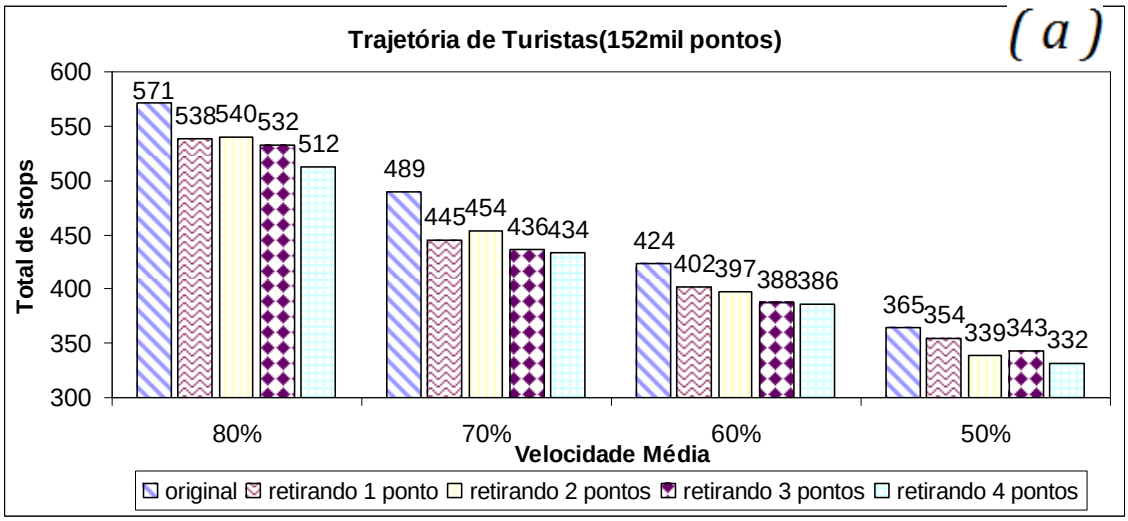
retirando 3 pontos	19 seg	84 seg	72%
retirando 4 pontos	15 seg	84 seg	83%
50% da original	37 seg	84 seg	66%
33% da original	24 seg	84 seg	71%
25% da original	18 seg	84 seg	78%
20% da original	15 seg	84 seg	82%

Tabela 3 - Porcentagem de otimização do algoritmo CB-SMOT conforme a retira de pontos.

4.2.2. Análise do total de stops gerados

Observando o Gráfico 5, é notório que, independente do tipo de limpeza utilizada, a quantidade de stops varia conforme aumenta a limpeza da trajetória original. Como era esperado, a variação no número de stops gerados neste conjunto de dados é maior.

Nota-se, observando os gráficos 5a e 5b que, assim como as trajetórias de carros, os resultados das trajetórias limpas, independente do tipo de limpeza, são diferentes em relação aos resultados das trajetórias originais. Porém, no conjunto de trajetórias de carros existem casos em que a soma total dos stops das tabelas limpas são maiores que a soma do total dos stops da original e neste outro conjunto de dados isso não ocorre. Mesmo assim, existem trajetórias limpas que geram mais ou menos stops que a trajetória original, podendo gerar vários resultados diferentes. No decorrer dessa seção serão explicados os resultados obtidos: quando não há alteração na quantidade de stops entre as trajetórias originais e limpas e quando as trajetórias limpas geram mais ou menos stops com relação aos stops da original.



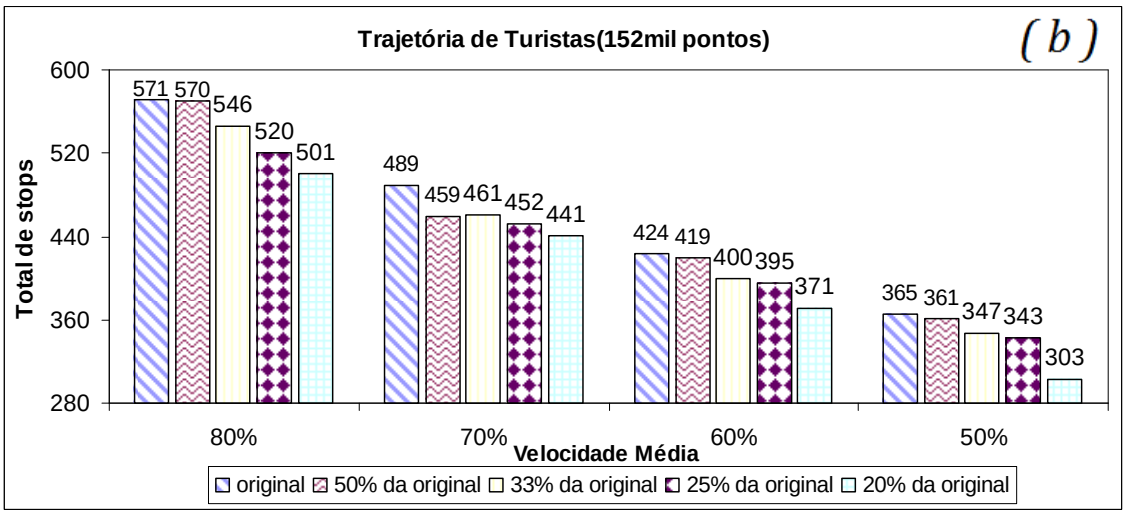


Gráfico 5 - Quantidade de stops por tabelas.

O resultado ideal seria não haver alteração entre os stops resultantes da execução do algoritmo sobre as tabelas originais e limpas. No entanto, devido à alteração da velocidade média da trajetória ocasionada pelo método de limpeza descrito acontece uma diferença. Um exemplo de limpeza e execução ideal do algoritmo pode ser observado na Figura 11, que mostra uma trajetória e os stops da trajetória original e das trajetórias limpas com a retirada de um até quatro pontos. Note que os stops não são exatamente iguais, sempre há uma pequena diferença, aceitável, devido à retirada dos pontos. Novamente, o mais importante é que os stops estão na mesma região e representam a mesma área e a mesma semântica.

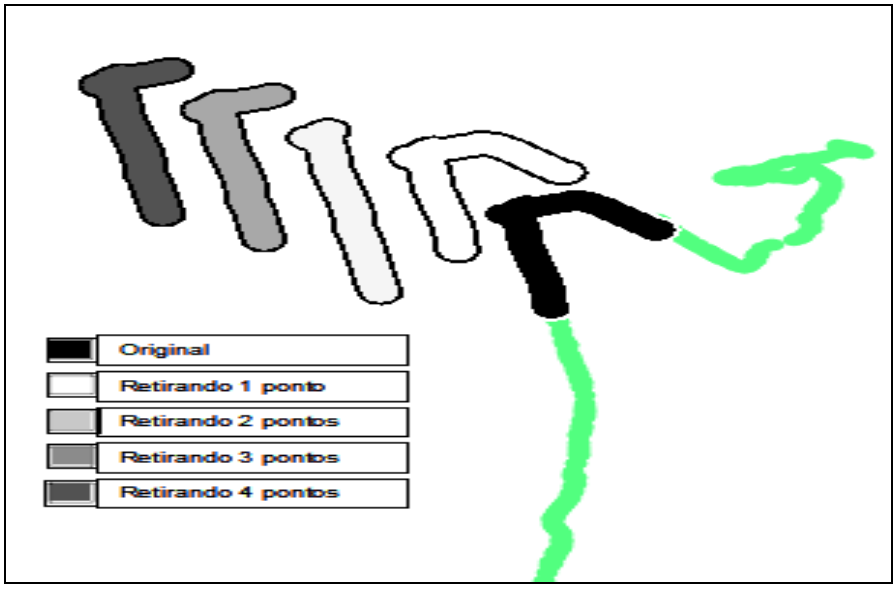


Figura 11 – Exemplo de limpeza com remoção de até 4 pontos em uma trajetória.

A grande diferença entre os resultados do conjunto de trajetórias de carros e de pedestres é observada quando os stops resultantes das trajetórias original e limpas são diferentes. Existem casos em que a remoção de pontos afeta a semântica dos stops resultantes e existem casos em que não afeta. Um dos casos em que a semântica não é afetada é explicado em seguida com a Figura 12. Já os casos em que a semântica é afetada será explicado posteriormente.

A Figura 12 mostra um caso em que os stops resultantes das trajetórias limpas geram menos stops que os originais. No entanto, a região dos stops originais é sempre representada pelos stops das trajetórias limpas, apesar de, em quantidade menor, ou seja, a semântica é preservada. Assim como no exemplo da Figura 9 do conjunto de trajetórias de carros, ocorre também a junção de dois stops próximos ou até mesmo sobrepostos da trajetória original, resultando em apenas um stop na trajetória limpa, considerando assim uma junção de dois stops originais.

Note que na Figura 12 dois stops da trajetória original foram aglutinados em apenas um stop nas trajetórias limpas. Note que os stops que se interceptam na trajetória original possuem o mesmo nome, uma vez que estes têm a mesma semântica. Assim, observa-se que ambos os stops caracterizam uma mesma região lenta. Em seguida, observe que os stops das trajetórias limpas também possuem o mesmo nome. Com isso, entende-se que não houve perda de informação, a semântica dos stops é preservada, embora sua geometria tenha sido alterada.

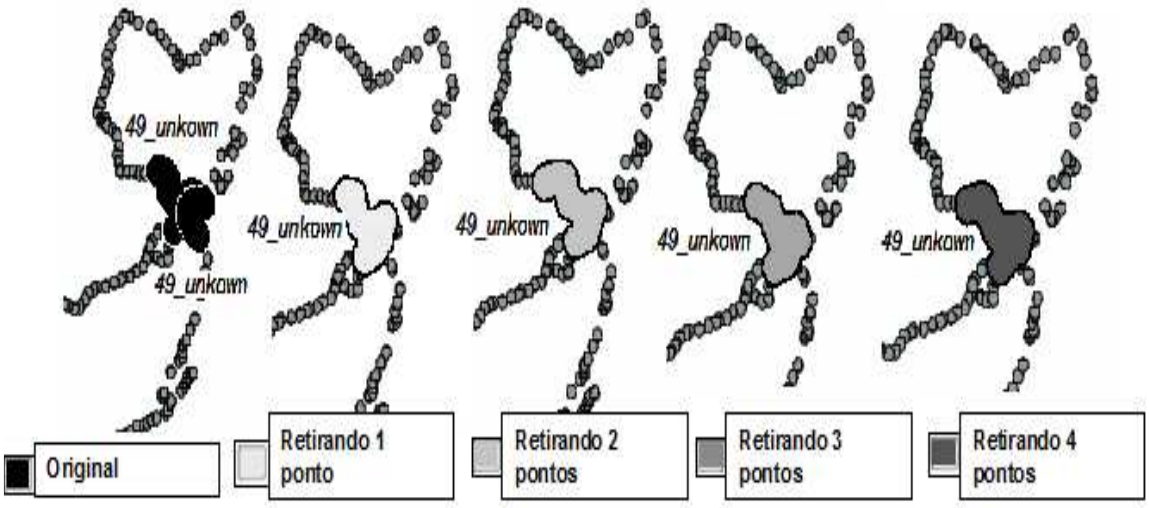


Figura 12 – Exemplo de limpeza e execução do algoritmo CB-SMOT onde os stops originais são em maior número que os stops das trajetórias limpas.

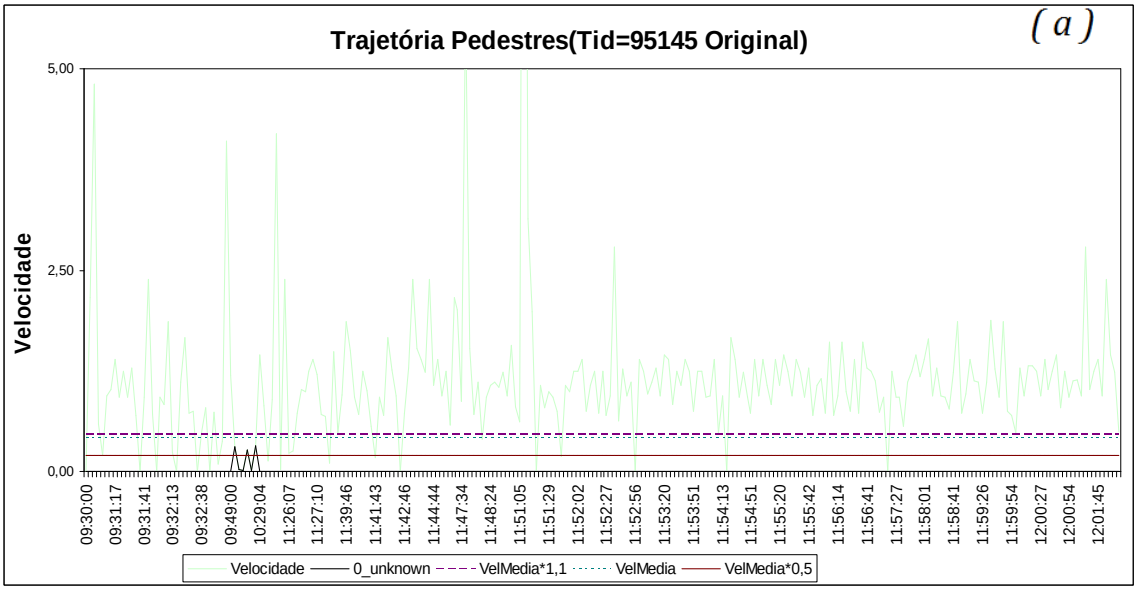
Observe o Gráfico 6 para analisar os casos em que a semântica foi afetada pela geração de stops que não eram gerados nas trajetórias originais. A primeira grande diferença entre os gráficos de pedestres e de carros é a quantidade de pontos, que pode ser observado nos gráficos 6 e 3. No Gráfico 3 do conjunto de trajetórias de carros nota-se que a retirada de até quatro pontos não afeta a densidade dos pontos do gráfico, os pontos continuam concentrados apesar da retirada de pontos pela limpeza das trajetórias. Já não se vê o mesmo resultado para o Gráfico 6 das trajetórias de pedestres, onde a trajetória original de pedestres possui uma quantidade de pontos muito inferior que a dos carros, devido a média de pontos criados por segundo. Para os carros, geralmente os aparelhos de GPS armazenam um ponto a cada segundo. Já para as trajetórias de pedestres, estes pontos estão sendo armazenados em média de 17 segundos, afetando consideravelmente o tamanho de cada trajetória de pedestres. Com isso, quando executada a limpeza, a retirada de um ou mais pontos pode criar uma região de baixa velocidade na trajetória, gerando assim novos stops.

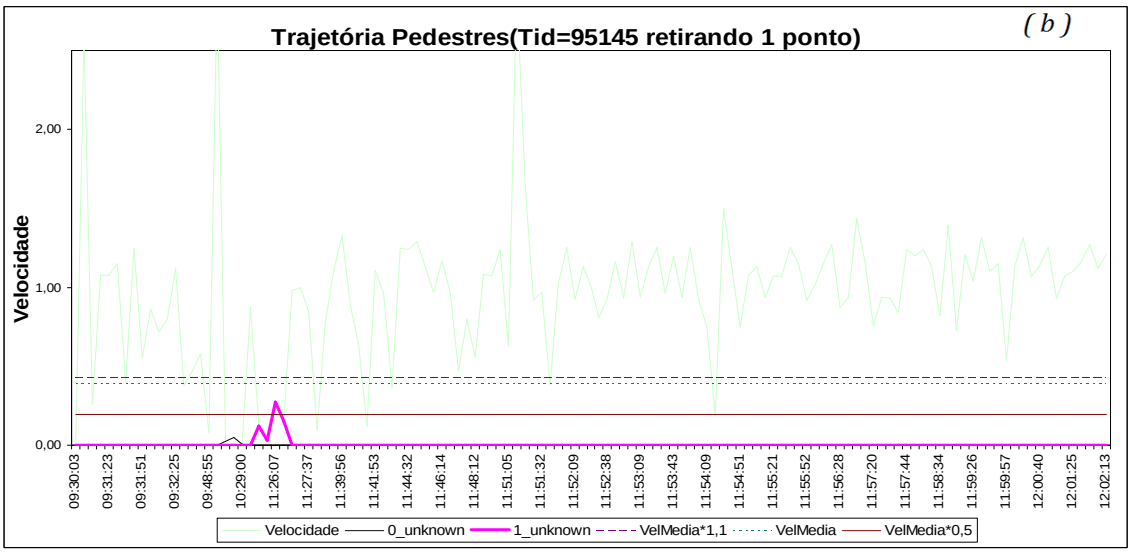
O Gráfico 6 foi gerado usando 0,5 de velocidade média, 600 segundos de tempo mínimo e 1,1 de limite de velocidade. O Gráfico 6 representa a trajetória 95145 e suas trajetórias limpas da tabela de trajetórias de pedestres de Amsterdam. Para as trajetórias limpas foi executada a técnica de amostragem sistemática com a remoção de um até quatro pontos. A trajetória original possui o total de 684 pontos. Já aplicando a técnica de amostragem sistemática removendo 1, 2, 3 e 4 pontos, gerou-se as trajetórias limpas contendo respectivamente, 341, 227, 169 e 136 pontos. Note que, mesmo aplicando a retirada de 4 pontos da trajetória de carro do Gráfico 3e, a quantidade de pontos ainda é maior que a trajetória original de pedestres deste exemplo. Com isso, já era esperado que os stops desta trajetória de pedestres fossem diferentes da original após a limpeza dos dados. Esta diferença é explicada a seguir.

No Gráfico 6, a primeira linha tracejada representa velocidade média multiplicada por 1,1 (parâmetro limite de velocidade informado). A segunda linha tracejada representa a velocidade média da trajetória e na terceira e última linha representa a velocidade média multiplicada por 0,5 (parâmetro velocidade média informada pelo usuário).

Para melhor entender o gráfico, foi colocado na legenda o nome do stop, assim facilita o entendimento da semântica associada. O Gráfico 6a que representa a trajetória original gerou apenas um stop (0_*unknown*) no período de 09:49:34 a 10:29:04. Para a

retirada de apenas um ponto (Gráfico 6b), o stop *0_unknown* da trajetória original foi preservado, mas com oscilações muito pequenas. Note que no período de 10:29:00 a 11:26:07 foi gerado outro stop(*1_unknown)* que não havia na trajetória original, devido a retirada de pontos que estavam nesse período, fazendo com que essa região ficasse com velocidade média abaixo de 0,5 da velocidade média e caracterizando um novo stop. Note que onde eram os dois stops do Gráfico 6b houve uma junção dos stops anteriores, ou seja, em todos os gráficos das trajetórias limpas a geografia e a semântica foram diferentes, pois regiões onde não eram consideradas como regiões de baixa velocidade, passaram a serem consideradas pela retirada de pontos.





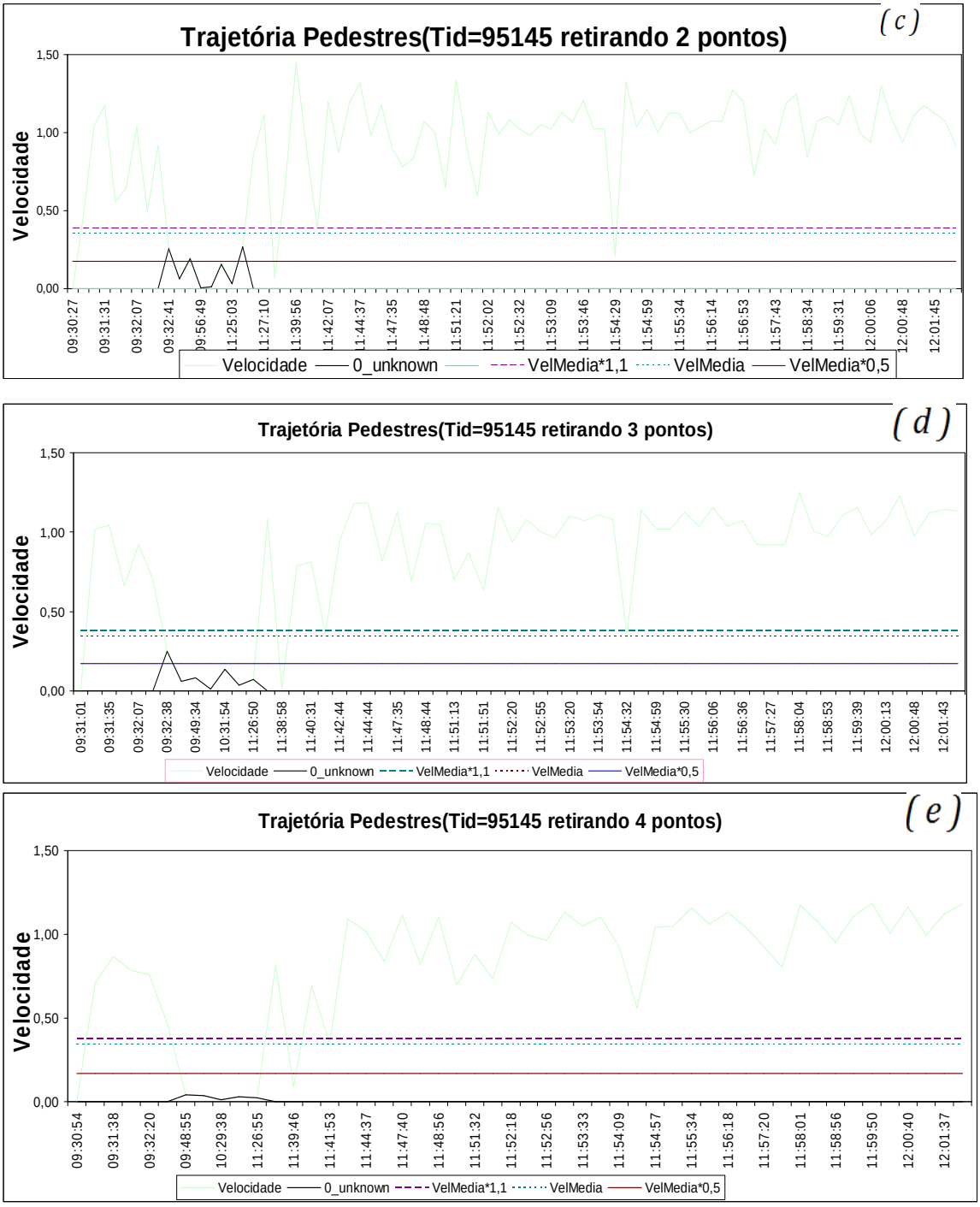


Gráfico 6 – Gráficos de velocidade onde mostra que a execução do algoritmo CB-SMOT criou mais stops e em locais diferentes.

Um outro exemplo de resultados divergentes entre os stops originais e limpos pode ser analisado na Figura 13 e no Gráfico 7. Ambos representam a trajetória 102993 das trajetórias de pedestres de Amsterdam. As trajetórias original, retirando 1, 2, 3 e 4 pontos são representadas, respectivamente, nas derivações ‘a’, ‘b’, ‘c’, ‘d’ e ‘e’ da Figura 13 e Gráfico 7. A trajetória original possui o total de 1099 pontos, já as trajetórias

limpas retirando 1, 2, 3 e 4 pontos possuem respectivamente 548, 364, 274 e 219 pontos.

O objetivo de mostrar as geometrias dos stops na Figura 13 é com a finalidade de visualizar o tamanho de cada stop e em que posição geográfica cada um foi gerado. Já o Gráfico 7 serve para mostrar quais e quantos pontos pertencem a cada stop e as velocidades dos pontos tanto na trajetória original quanto na trajetória limpa. Uma das características desta trajetória de pedestres é o tamanho de cada stop. Observe que na trajetória original os três stops são gerados, respectivamente, com 2, 4 e 6 pontos. Sendo assim, já é sabido que, se for aplicada uma técnica de retirada de pontos e alguns desses pontos forem removidos aumenta a chance deste stop não ser gerado na trajetória limpa.

Neste exemplo pode-se observar que alguns stops foram criados e outros foram retirados depois da limpeza da trajetória original. Isto acontece pelo mesmo motivo do Gráfico 6 onde apareceu um novo stop. O problema da quantidade de dados também aparece para esta trajetória de pedestres onde no Gráfico 7 a quantidade pontos diminui conforme a retirada de pontos e acaba gerando ou retirando uma região de velocidade média baixa, assim as trajetórias limpas geram mais ou menos stops.

Observe que os stops limpos que mais se parecem com os originais estão na Figura 13c e no Gráfico 7c, onde foi aplicada uma retirada de dois pontos. Nota-se que a quantidade de stops foi equivalente e ambos os stops resultantes estão na mesma região, a única diferença é o tamanho dos stops, mesmo sendo assim um bom resultado. Este resultado foi obtido porque a remoção de pontos não eliminou os pontos dos quais fazem parte dos stops da trajetória original, sendo assim as regiões de baixa velocidade foram mantidas na trajetória limpa.

Comparando a trajetória na qual foi retirado um ponto pela limpeza (Figura 13b) com a original (Figura 13a), pode-se observar que a quantidade de stops diminuiu. Observando o Gráfico 7b pode-se notar que os pontos próximos ao tempo 08:52:43 foram removidos pela técnica de limpeza, confirmando o foi dito anteriormente, o stop removido na trajetória limpa era composto de apenas dois pontos, e depois da técnica de limpeza um desses pontos foi removido. Como os pontos vizinhos possuem uma velocidade alta, fizeram com que a média da velocidade dessa região ficasse mais alta que a velocidade média da trajetória, assim não gerou o stop. O mesmo acontece para o stop do período de tempo de 10:08:41 onde na trajetória retirando 4 pontos(Figura 13e e Gráfico 7e) esse stop não foi gerado, uma vez que os pontos que eram de baixa velocidade foram excluídos.

O último caso a ser analisado nessa trajetória de pedestres é o stop que foi gerado a mais no Gráfico 7d e na Figura 13d, onde foi utilizada uma remoção de 3 pontos. Nota-se que uma região de baixa velocidade foi criada pela remoção dos pontos que passavam do limite de velocidade do período de 08:53:03 até 09:23:10. Estes pontos dividiam dois potenciais stops de baixa velocidade que não possuíam a duração mínima de tempo. Posteriormente a limpeza, ambos os potenciais stops se juntaram em apenas um stop com duração suficiente, justificando a criação de um stop a mais na trajetória limpa. Assim sendo, já era esperado que a variação do número de stops fosse grande para este conjunto de dados. No entanto, para algumas aplicações essa redução de dados pode ser interessante, já que a eliminação de um ponto de alta velocidade dentro de uma região de baixa velocidade poderia ser um pequeno ruído que impedia a geração de um stop na trajetória original, mas que com limpeza permitiu a geração do stop.

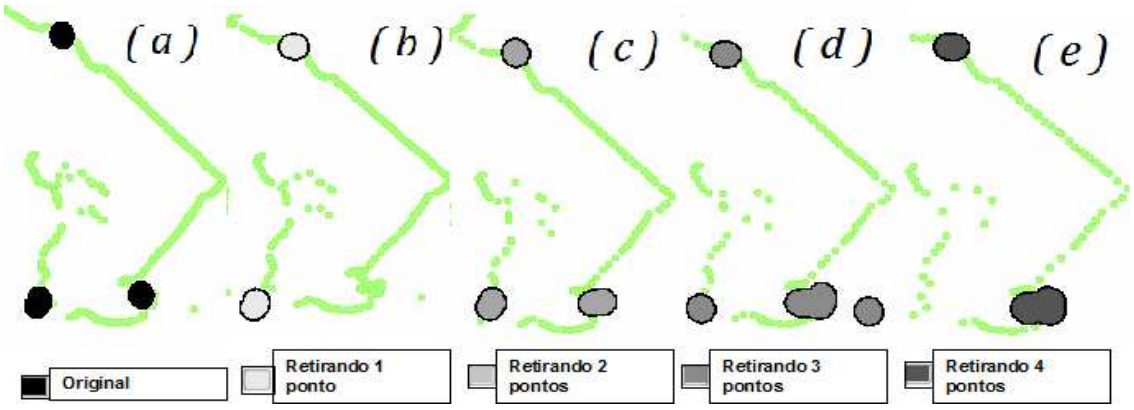
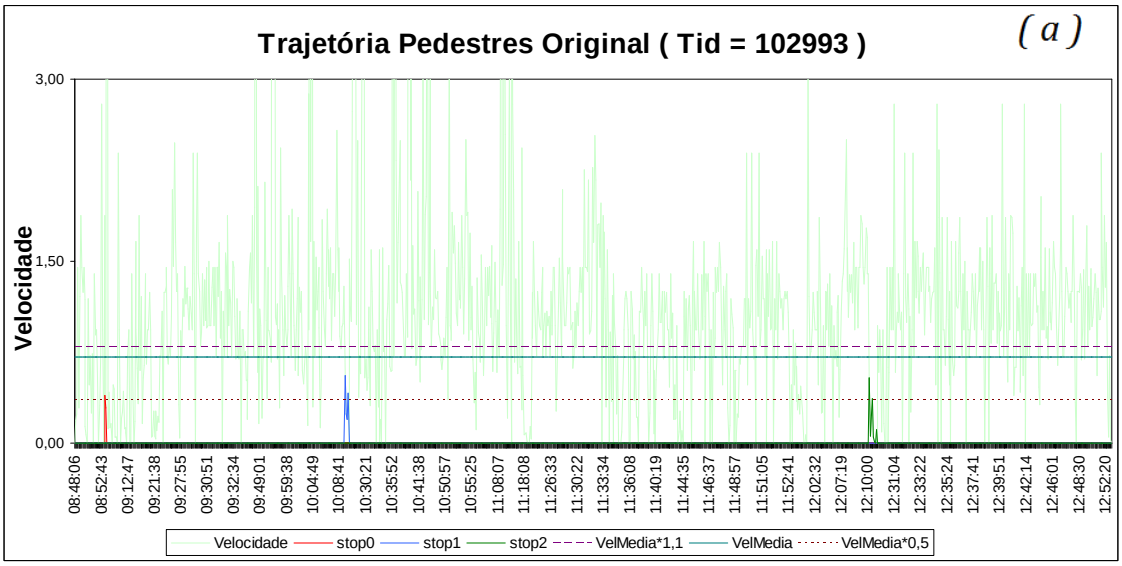
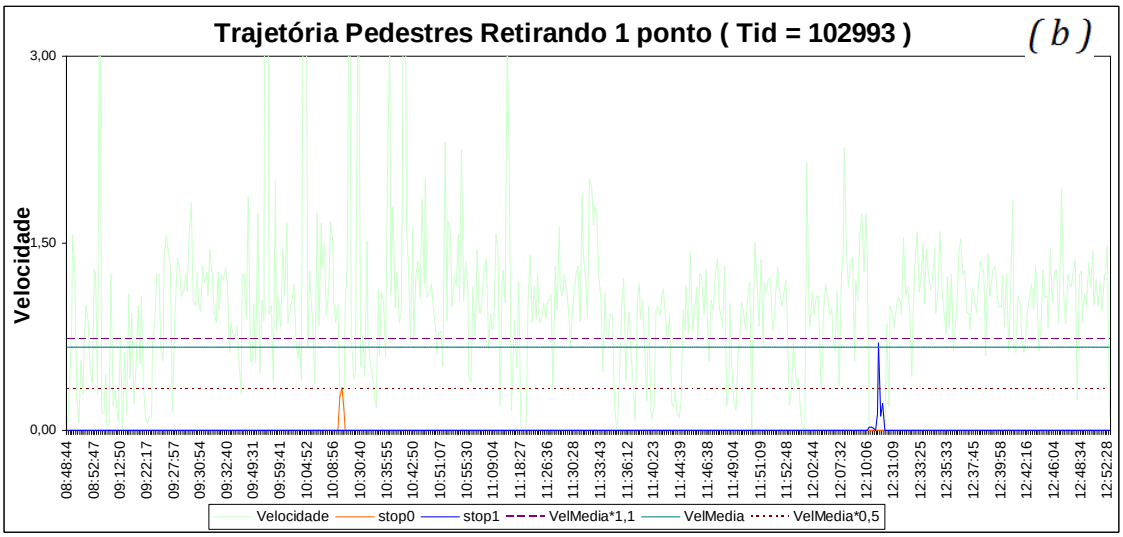
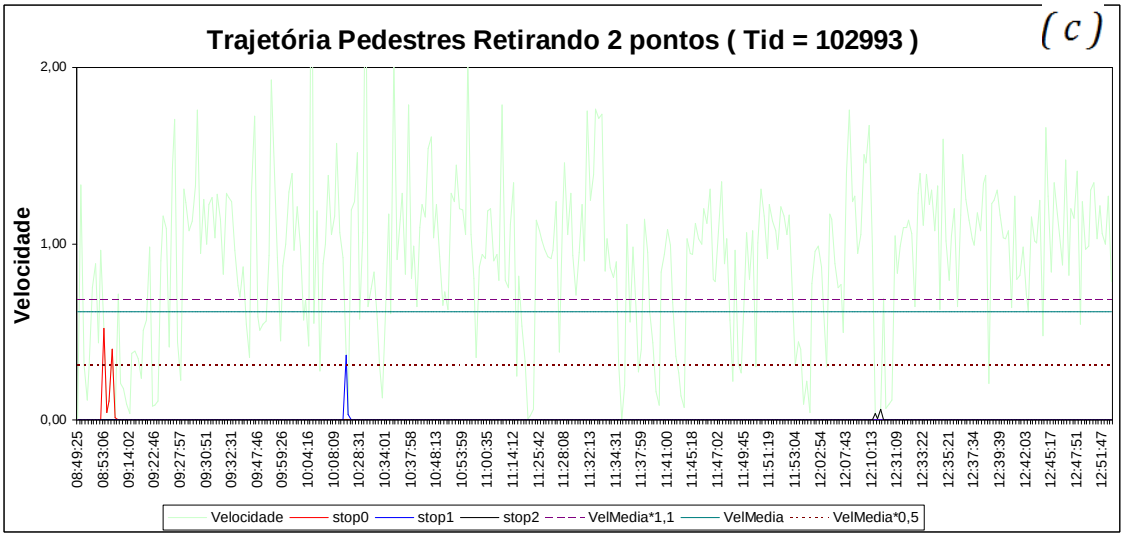
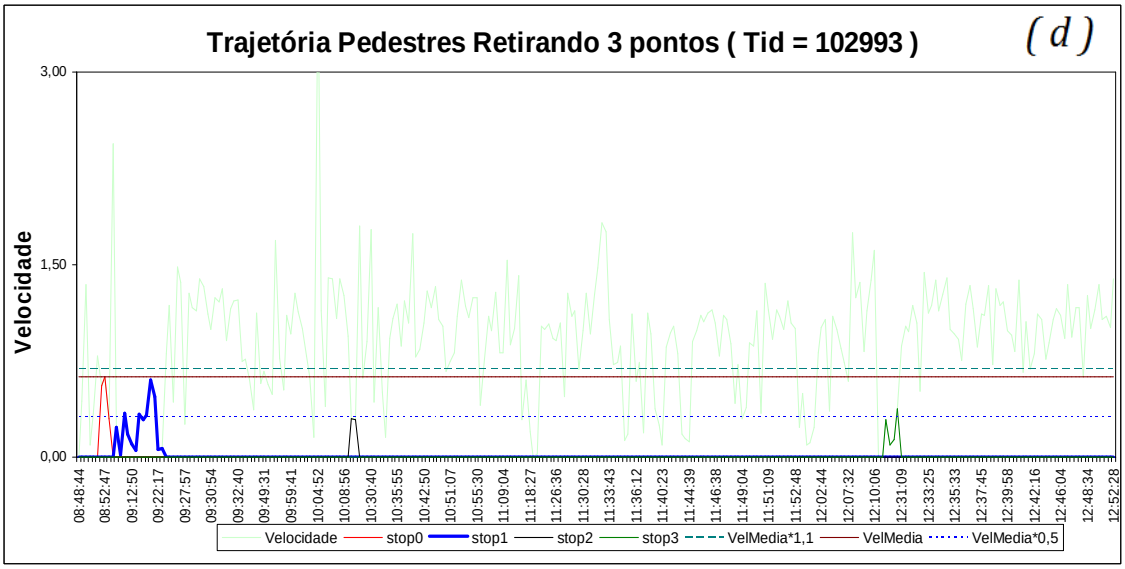


Figura 13 – Exemplo de limpeza e execução do algoritmo CB-SMOT onde os stops originais são diferentes dos stops das trajetórias limpas.




Trajetória Pedestres Retirando 1 ponto (Tid = 102993)
(b)
Velocidade
3,00
1,50
0,00
08:48:44 08:52:47 09:12:50 09:22:17 09:27:57 09:30:54 09:32:40 09:49:31 09:59:41 10:04:52 10:08:56 10:30:40 10:35:55 10:42:50 10:51:07 10:55:30 11:09:04 11:18:27 11:26:36 11:30:28 11:33:43 11:36:12 11:40:23 11:44:39 11:46:38 11:49:04 11:51:09 11:52:48 12:02:44 12:07:32 12:10:06 12:31:09 12:33:25 12:35:33 12:37:45 12:39:58 12:42:16 12:46:04 12:48:34 12:52:28
Velocidade stop0 stop1 VelMedia*1,1 VelMedia VelMedia*0,5


Trajetória Pedestres Retirando 2 pontos (Tid = 102993)
(c)
Velocidade
2,00
1,00
0,00
08:49:25 08:53:06 09:14:02 09:22:46 09:27:57 09:30:51 09:32:31 09:47:46 09:59:26 10:04:16 10:08:09 10:28:31 10:34:01 10:37:58 10:48:13 10:53:59 11:00:35 11:14:12 11:25:42 11:28:08 11:32:13 11:34:31 11:37:59 11:41:00 11:45:18 11:47:02 11:49:45 11:51:19 11:53:04 12:02:54 12:07:43 12:10:13 12:31:09 12:33:22 12:35:21 12:37:34 12:39:39 12:42:03 12:45:17 12:47:51 12:51:47
Velocidade stop0 stop1 stop2 VelMedia*1,1 VelMedia VelMedia*0,5


Trajetória Pedestres Retirando 3 pontos (Tid = 102993)
(d)
Velocidade
3,00
1,50
0,00
08:48:44 08:52:47 09:12:50 09:22:17 09:27:57 09:30:54 09:32:40 09:49:31 09:59:41 10:04:52 10:08:56 10:30:40 10:35:55 10:42:50 10:51:07 10:55:30 11:09:04 11:18:27 11:26:36 11:30:28 11:33:43 11:36:12 11:40:23 11:44:39 11:46:38 11:49:04 11:51:09 11:52:48 12:02:44 12:07:32 12:10:06 12:31:09 12:33:25 12:35:33 12:37:45 12:39:58 12:42:16 12:46:04 12:48:34 12:52:28
Velocidade stop0 stop1 stop2 stop3 VelMedia*1,1 VelMedia VelMedia*0,5

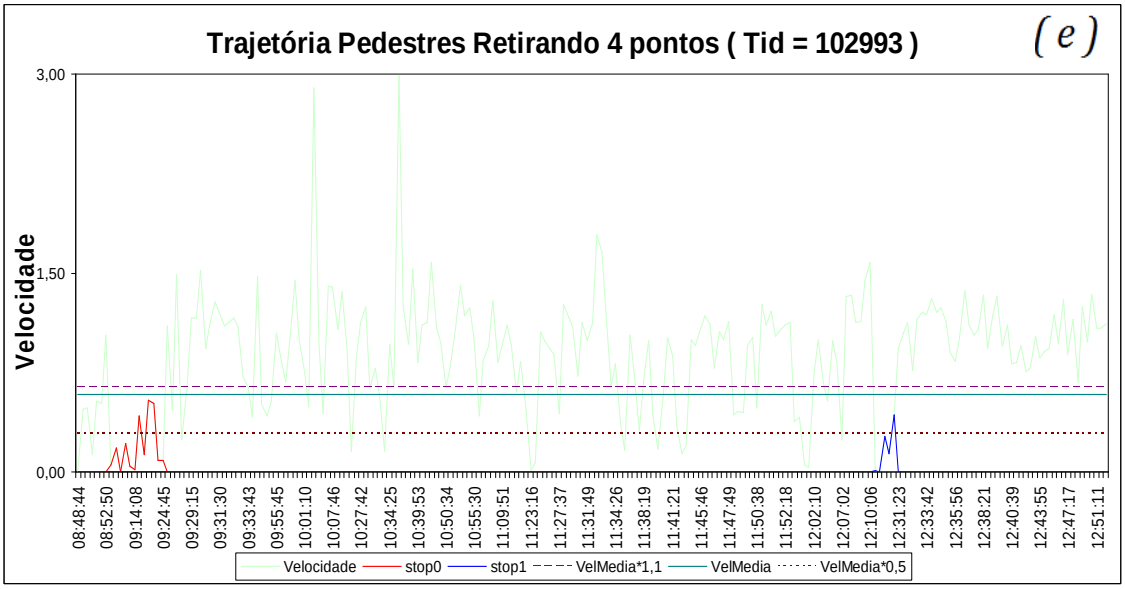


Gráfico 7 – Gráficos de velocidade onde mostra que a execução do algoritmo CB-SMOT criou mais e menos stops em locais diferentes.

4.3 Discussão dos Resultados

Nesta seção são descritas algumas conclusões importantes sobre os experimentos. A redução de dados pode afetar o resultado do algoritmo CB_SMOT em apenas duas formas. Uma acontece quando a redução de dados afeta a velocidade média da trajetória e quando a densidade dos pontos é baixa, conforme visto no conjunto de dados de pedestres, onde novas regiões de baixa velocidade são criadas pela limpeza dos dados, podendo dar origem a novos e mais stops.

A Figura 14 ilustra como a redução de dados pode afetar a velocidade média da trajetória. Sabe-se que a velocidade média da trajetória é a base do calculo do algoritmo CB_SMOT. A velocidade média é calculada pela soma da distância entre dois pontos, dividida pela diferença do tempo entre os dois pontos. Imagine então uma trajetória com os pontos mostrados da Figura 14. Observe a distância entre os pontos A, B, C, D e E. Agora imagine que na remoção de 1 ponto, os pontos B e D desapareçam da trajetória. Logo, é possível notar que a duração da trajetória permaneceu a mesma, porém a distância entre A e C é muito menor do que a distância entre A e C quando B fazia parte da trajetória. Da mesma forma, a distância entre C é E é inferior a distância entre C, D e E. Como conseqüência, a velocidade média da trajetória é afetada explicando a variação no número de stops.

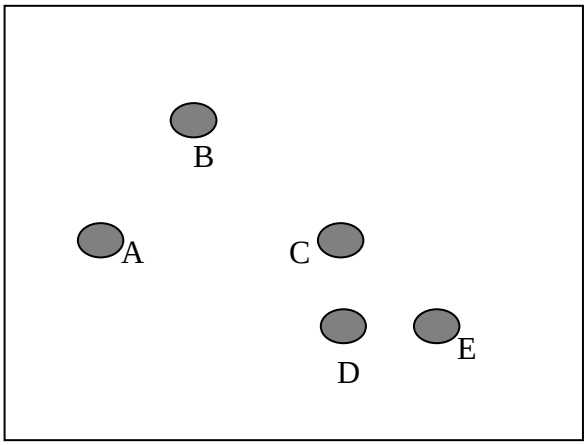


Figura 14 – Exemplo do cálculo da velocidade média para trajetória original e limpa.

Uma outra análise que convém mencionar é quanto ao tempo gasto para eliminar os pontos das trajetórias durante a limpeza. Logicamente, o tempo gasto para fazer essa limpeza está diretamente associado à quantidade de dados de trajetórias. A limpeza das trajetórias de carros removendo 1, 2, 3, e 4 levou respectivamente “3,8” , “3,73”, “3,64” e “3,52” minutos .

O tempo de remoção de 50%, 33%, 25% e 20% dos dados levou respectivamente “3,67”, “3,47”, “3,25” e “3,20” minutos.

Embora a limpeza de dados consuma algum tempo, é importante enfatizar que a limpeza dos dados, independentemente da técnica utilizada, é realizada uma única vez, enquanto algoritmos de mineração de dados e adição de semântica em trajetórias como o CB-SMOT são executados dezenas de vezes, pelo fato do processo de mineração de dados ser iterativo e interativo. Além disso, o processo de mineração é repetido porque não é na primeira execução que o usuário encontra padrões e resultados desejados. Mineração de dados não é um processo trivial.

5. CONCLUSÃO E TRABALHOS FUTUROS

Com a popularização e acessibilidade de aparelhos de coleta de informações geográficas como aparelhos de telefone celular, GPS ou qualquer outro dispositivo móvel aumentou significativamente disponibilidade de bases de dados espaço-temporais. Essas bases de dados são formadas por conjuntos de trajetórias de objetos móveis. Essas trajetórias são chamadas de trajetória brutas, pois não possuem contexto ou semântica associada. Com isso, foram criados alguns métodos de análise e extração de semântica sobre essas trajetórias brutas, sendo essa nova trajetória chamada trajetória semântica. No entanto, os tempos de processamento desses métodos são muito altos, podendo chegar a mais de 6 horas dependendo do volume de dados. Para isso, este trabalho vem com o objetivo de otimização deste tempo de processamento.

Como forma de otimização desse tempo de processamento, optou-se pela aplicação de técnicas de amostragem estatística sobre as trajetórias brutas antes da aplicação do algoritmo, com isso, tem-se que as trajetórias brutas ficam menores e mais limpas. Assim, gerando a hipótese de que se a quantidade de dados a ser analisada pelo algoritmo for menor, conseqüentemente, o tempo de processamento vai ser menor. Para isso, foram realizados experimentos sobre dois tipos de trajetórias diferentes: trajetórias de carros e de pedestres.

O objetivo geral das técnicas de redução de dados foram satisfatórias em relação ao tempo de processamento tanto para as trajetórias de carros como de pedestres, validando a hipótese anteriormente citada. Notou-se que, nos experimentos, houve uma otimização do tempo de processamento de no mínimo 55% utilizando a técnica de amostragem sistemática retirando apenas um ponto entre cada 2 pontos da trajetória original. Esta porcentagem se mantém igual para a técnica de amostragem aleatória simples com a utilização de 50% dos pontos da trajetória original.

Apesar de o tempo de processamento ter sido positivo independente da retirada de dados e dos tipos de trajetórias, os resultados do algoritmo foram diferentes do esperado. Ocorreram dois resultados diferentes e estes estão associados ao tipo de dado utilizado. Para as trajetórias de carros a aplicação das técnicas de retirada de dados acarreta uma pequena diferença no número de stops e seu formato, mas essas diferenças não afetam a semântica dos stops resultantes comparados com os stops originais. Neste sentido, o resultado foi satisfatório para as trajetórias de carros.

Já para as trajetórias de pedestres, existiram também alguns resultados satisfatórios. No entanto, nos casos em que os stops resultantes foram diferentes, a semântica dos stops foi alterada. Isso ocorreu unicamente pela diferença da densidade dos pontos da trajetória de carros e pedestres. Para as trajetórias de carros os dados geralmente são coletados a cada segundo, independente do carro estar em movimento ou não. Com isso, em apenas poucos minutos uma trajetória possui milhares de pontos. O mesmo não acontece para as trajetórias de pedestres onde a média de tempo entre um ponto e outro é de 17 segundos, sendo que em muitos casos essa diferença não é constante. Assim, para qualquer uma das técnicas de redução de dados os resultados dos stops podem ser diferentes, criando ou retirando com facilidade uma região de baixa velocidade, em conseqüência alterando a semântica dos resultados. Mesmo alterando a semântica do stop, este resultado ainda pode ser interessante. Por exemplo, se em uma região a eliminação de um ponto de ruído causou a geração de um stop que não era gerado na trajetória original, este stop é uma informação importante para o usuário, que pelo fato de ter sido gerada, vai permitir que ele uma nova análise.

Este trabalho também identificou uma série de erros e problemas na implementação do algoritmo CB_SMOT que atrasaram a conclusão deste trabalho. Por exemplo, no cálculo da velocidade de cada ponto, a distância estava sendo dividida pelo tempo em milissegundos. Já no cálculo da velocidade média da trajetória, o tempo era calculado em segundos. Ou seja, quando executava o algoritmo CB_SMOT, no primeiro passo, todos os pontos possuíam a velocidade menor que o parâmetro velocidade média da trajetória devido a diferença entre essas velocidades serem de 10^3 casas decimais, assim todos os stops potenciais que possuíam o mínimo de tempo informado pelo usuário eram criados como stop, mesmo que a velocidade de alguns desses pontos ultrapassassem o limite de tempo. A versão utilizada para os experimentos foi com esse problema solucionado, porém nos fez reescrever todos os gráficos e análises dos stops.

Como trabalhos futuros, outras técnicas de redução de dados podem ser estudadas.

6. REFERÊNCIAS

[Alvares 2007] Alvares, L. O.; Bogorny, V.; Kuijpers, B.; Macedo, J. A. F.; Moelans, B.; Vaisman, A.: A Model for Enriching Trajectories with Semantic Geographical Information.. In: Proc. of the ACM 15th International Symposium on Advances in Geographic Information Systems (ACM-GIS'07), Seattle, Washingthon, 7-9 November (2007).pp. 162-169

[Álvares 2010] Alvares, Luis O. ; Palma, Andrey ; Oliveira, G. ; Bogorny, V. Weka-STPM: from trajectory samples to semantic trajectories. In: Workshop de Sofware Livre, 2010, Porto Alegre. WSL, 2010.

[Barbeta 2001] Barbeta, Pedro Allberto: *Estatística Aplicada ás Ciências Sociais.* 4° Edição, Editora UFSC, Florianópolis, SC, 2001. pp. 41-58.

[Bogorny 2006] Bogorny, V.; Palma, A.; Engel, P.; Alvares, L.O.: *Weka-GDPM: Integrating Classical Data Mining Toolkit to Geographic Information Systems*. In: SBBD Workshop on Data Mining Algorithms and Aplications(WAAMD'06), Florianópolis, Brazil, October 16-20, (2006). pp.9-16

[Frank 2005] Frank, E., Hall, M. A., Holmes, G., Kirkby, R., Pfahringer, B., Witten, I. H., and Trigg, L. (2005). Weka - a machine learning workbench for data mining. In Maimon, O. and Rokach, L., editors, The Data Mining and Knowledge Discovery Handbook, pages 1305–1314. Springer.

[Lavado 2001] Lavado EL, Castro AA. Projeto de pesquisa (Parte V – amostra). In: Castro AA. Planejamento da Pesquisa. São Paulo: AAC; 2001. Disponível em URL: Projeto de pesquisa; Acessado em agosto de 2010.

[Manso 2010] Manso, J. A; Times, V. C.; Oliveira, G.; Alvares, L.O.; Bogorny, V. DB-SMoT: a Direction-based spatio-temporal clustering method. Fifth IEEE International Conference on Intelligent Systems (IEEE IS 2010), 2010.

[OGC 2008] Open Geospatial Consortium.
Disponível em URL: http://www.ogc.gov.uk/news_2008.asp
Acessada em agosto de 2010.

[Palma 2008] Palma, A. T; Bogorny, V.; Kuijpers, B.; Alvares, L.O. A Clustering-based Approach for Discovering Interesting Places in Trajectories. In: 23rd Annual Symposium on Applied Computing, (ACM-SAC'08), Fortaleza, Ceara, 16-20 March (2008) Brazil. pp. 863-868.

[Pelekis 2008] Pelekis, N.; Marketos, G.; Frentzos, E.; Ntoutsi, I.; Raffaetà, A.; Theodoridis, T.; "Building Real-World Trajectory Warehouses" In: 7th International ACM Workshop on Data Engineering for Wireless and Mobile Access, Vancouver, Canada, 2008, pp. 8–15.

[Quantum 2010] Documentação sobre o aplicativo Quantum Gis, versão 1.4.0.
Disponível em URL: http://www.qgis.org/pt/documentation/manuals.html
Acessada em agosto de 2010.

[Spaccapietra 2008] Spaccapietra, S.; Parent, C.; Damiani, M.L.; Macedo, J. A. F. de ; Porto, F.; Vangenot, C.; A conceptual view on trajectories, Data & Kownledge Egineering, 65 n.1, p126-146, April 2008.

[Unama2010] Documentação do curso de graduação de estatística.
Disponível em URL:
http://arquivos.unama.br/nead/gol/gol_adm_2mod/estatistica/pdf/ESTA_impresso_aula07.pdf
Acessada em agosto de 2010.

[Weka 2009] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, Ian H. Witten (2009); The WEKA Data Mining Software: An Update; SIGKDD Explorations, Volume 11, Issue 1.

Anexo 1 – Artigo.

Análise da Redução de Dados em Trajetórias de Objetos Móveis

Hércules M. Avancini[1]

[1] Departamento de Informática e Estatística – Universidade Federal de Santa Catarina
Florianópolis – SC – Brasil

hercules@inf.ufsc.br

Abstract. *With the decreasing price of equipments for collecting geographic information, the availability of geographic databases increased significantly. These databases store what we call raw trajectories of moving objects. However, these trajectories have no context or semantics associated to the original data. Therefore, some algorithms have been developed to add semantic information to trajectories. The processing time of these algorithms, however, is high, reaching more than six hours depending on the size of the database. To reduce this problem, this work aims at optimizating processing time through the application of statistical sampling techniques to reduce the volume of raw data of trajectories before running the algorithms. We applied these techniques on two different types of trajectories and, subsequently, we evaluated the processing time and the quality of the output data.*

Resumo. *Com a redução dos preços de aparelhos de coleta de informações geográficas aumentou significativamente a disponibilidade de bases de dados geográficas. Essas bases de dados são formadas por conjuntos de trajetórias brutas de objetos móveis. No entanto, essas trajetórias não possuem contexto ou semântica associada. Por isso foram criados alguns algoritmos para adicionar semântica a essas trajetórias. No entanto, os tempos de processamento desses algoritmos são elevados, podendo chegar a mais de 6 horas dependendo do volume de dados. Para isso, este trabalho vem com o objetivo de otimização deste tempo de processamento, através da aplicação de técnicas de amostragem estatística para reduzir o volume de dados das trajetórias brutas antes da aplicação do algoritmo, reduzindo assim o tempo de processamento. Técnicas foram aplicadas sobre dois tipos de trajetórias diferentes e, posteriormente, foi analisado o tempo de processamento que se consegue otimizar e se a redução de dados afeta os resultados do algoritmo.*

1. Introdução

A redução nos preços de aparelhos de coleta de informações geográficas como aparelhos de Sistema de Posicionamento Global (GPS), de telefone celular ou qualquer outro dispositivo móvel aumentou significativamente a utilização de serviços de georeferenciamento por todos os segmentos da população. Inúmeras bases de dados geográficas estão sendo geradas com grandes volumes de dados. Essas bases são formadas por conjuntos de percursos percorridos por um corpo em um determinado período de tempo. Este percurso pode ser chamado de trajetória.

Uma trajetória consiste no caminho percorrido por um dado objeto no espaço durante um movimento. Para o escopo deste trabalho, uma trajetória é um conjunto de pontos no espaço (x,y) ordenados no tempo (t). Ou seja, podemos definir uma trajetória como um conjunto de pontos (x,y,t) .

Com isso, podemos definir que um objeto móvel é qualquer objeto cuja posição geográfica muda continuamente com o passar do tempo, como por exemplo quando uma pessoa caminha, mesmo que em distâncias curtas, em um shopping center. As trajetórias desses objetos móveis atualmente estão sendo utilizadas em análises de uma série de problemas atuais e relevantes, tais como o caos do transporte nos grandes centros, desastres climáticos, monitoramento de criminosos e a transmissão de doenças através da migração de aves.

Os aparelhos de GPS são exemplos de dispositivos que geram a referência espacial de um objeto no decorrer do tempo. A maioria destes dispositivos geram grandes quantidades de dados com informações redundantes, uma vez que por default coletam um ponto a cada segundo para a trajetória. Assim, quanto menor o intervalo de tempo de coleta, entre dois pontos, maiores serão as trajetórias e as bases de dados.

As trajetórias brutas na forma como são criadas contém poucas informações relevantes, basicamente representando pontos no tempo e no espaço. Por isso, trabalhos acadêmicos relativos à análise e o enriquecimento semântico destas trajetórias vêm sendo desenvolvidos com o intuito de descobrir conhecimento sobre toda ou parte das mesmas.

Recentemente, Spaccapietra [Spaccapietra 2008] introduziu um novo modelo de raciocínio sobre trajetórias, que nos permite uma poderosa análise semântica, baseado nos conceitos de stops e moves. O primeiro é um local de parada do objeto por um determinado período de tempo. Já o segundo representa os pontos da subtrajetória que estão entre os stops.

Todos os stops precisam ser calculados, e o método varia de acordo com a aplicação e o interesse do usuário. Nosso grupo de pesquisa já desenvolveu três métodos para este fim, IB-SMOT [Alvares 2007], CB-SMOT [Palma2008] e DB-SMOT [Manso2010]. O primeiro calcula stops com base na intersecção da trajetória com objetos geográficos de interesse. O segundo calcula stops em lugares onde a velocidade do objeto é baixa e o terceiro gera stops onde a direção da trajetória varia mais que um determinado valor. Para todos os métodos, os stops são gerados quando o objeto permaneceu no stop por um período mínimo de tempo.

Considerando uma amostra de dados real de veículos que circulam na cidade do Rio de Janeiro, uma tabela de trajetórias do banco de dados tem 7 milhões de pontos e 2 mil trajetórias, podendo levar mais de 6 horas para realizar uma análise de enriquecimento semântico destas trajetórias para gerar os stops e moves. Com isso, vê-se a necessidade da otimização destes algoritmos ou a redução do volume de dados para a diminuição do tempo de processamento.

O principal objetivo deste trabalho é diminuir consideravelmente o tempo de processamento destes algoritmos de enriquecimento semântico de trajetórias. Para isto será efetuada, antes da execução do algoritmo, uma limpeza nos dados de trajetórias através de algumas técnicas de redução de dados e amostragens.

O restante deste trabalho é organizado da seguinte forma: O Capítulo 2 apresenta alguns conceitos de trajetórias brutas e semânticas. No Capítulo 3 são abordados os métodos de redução de dados e uma breve descrição da implementação destes métodos. No Capítulo 4 são apresentados experimentos e finalmente o Capítulo 5 conclui o trabalho e comenta sobre possíveis trabalhos futuros.

2. Conceitos básicos

Para entendermos como serão aplicadas as técnicas de redução e limpeza dos dados de trajetórias de objetos móveis alguns conceitos sobre trajetórias brutas e trajetórias semânticas serão apresentados. Alguns conceitos básicos sobre o algoritmo

CB-SMOT também serão apresentados, o qual foi utilizado para avaliar os resultados experimentais deste trabalho.

2.1. Trajetórias

Como citado anteriormente, trajetórias de objetos móveis podem ser consideradas como o caminho seguido por um objeto em movimento no tempo e no espaço. Para este escopo, uma trajetória é um conjunto de pontos no espaço (x,y) ordenados pelo tempo (t). Cada ponto em uma trajetória representa uma posição no espaço em um determinado instante de tempo.

Uma pessoa com qualquer aparelho de georeferenciamento pode se movimentar de várias formas, gerando assim vários tipos de trajetórias diferentes. Por exemplo, a trajetória de uma viagem de avião, de carro ou a pé tem características completamente diferentes. Essas trajetórias somadas em período de tempo formam um grande volume de dados. No entanto, essas trajetórias geralmente são apenas dados armazenados, sem informação adicional sem qualquer contexto ou semântica, sendo posteriormente um tipo de trajetória muito difícil de analisar, entender e extrair informações interessantes. Esse tipo de trajetória é chamado de trajetória bruta.

Definição 1. *Trajetória bruta*: Uma trajetória bruta é uma lista de pontos espaço-temporal $\{ p_0= (x_0, y_0, t_0), p_1= (x_1, y_1, t_1),... p_n=(x_n, y_n, t_n)\}$, onde $x_i,y_i \in \Re$, $t_i \in \Re^+$, para $i = 0,1,2,...,n$ e $t_0<t_1<t_2...<t_n$.

Trajetórias brutas são apenas seqüências de pontos espaciais ordenados pelo tempo de ocorrência de cada ponto. A Figura 1 mostra uma trajetória bruta ao longo do tempo.

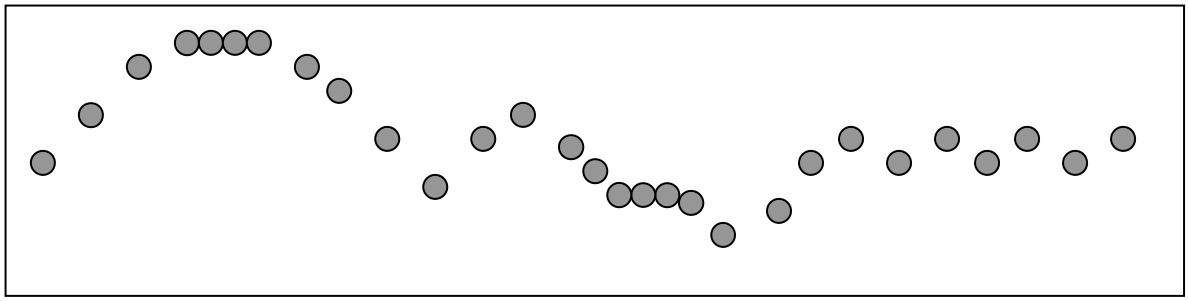


Figura 1 - Trajetória bruta representado por diversos pontos no espaço.

2.2. Trajetórias semânticas

Tendo em vista a desvantagem referente às trajetórias brutas citadas anteriormente, as trajetórias semânticas apresentam a vantagem de serem melhor compreendidas, pois além dos pontos ou dados brutos da trajetória são apresentadas informações adicionais da trajetória ou da região onde ela está inserida. A Figura 2 apresenta a trajetória de um turista enriquecida semanticamente com os lugares que o turista visitou.

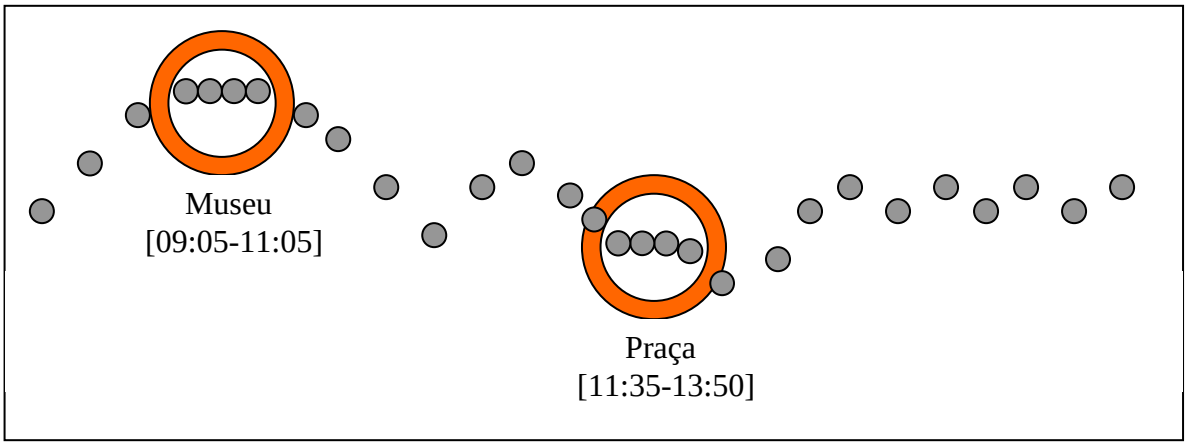


Figura 2 - Trajetória semântica, focando os movimentos e os pontos de parada de um turista

Algumas informações semânticas podem ser retiradas da Figura 2, como o que o turista desta trajetória fez entre as 08:30 da manhã até as 14:40. Já se compararmos várias trajetórias, de vários turistas, pode-se saber, por exemplo, quais pontos turísticos são mais procurados, ou quais conjuntos de pontos que são visitados primeiro. Essas informações podem ser usadas para mineração de dados e obterem-se várias conclusões sobre o contexto das trajetórias analisadas.

As trajetórias semânticas são obtidas através de métodos inteligentes. Alvares [ALVARES 2007] cita que para muitos domínios de aplicação, informações úteis só podem ser extraídas a partir de dados de trajetórias se alguma semântica ou informação geográfica forem considerados. Uma das formas que tem sido utilizada para enriquecer as trajetórias com informações semânticas é através do método stops e moves, proposto por Spaccapietra [Spaccapietra 2008].

Os *stops* representam a parte mais relevante de uma trajetória para uma aplicação, considerando que o objeto tenha permanecido um período mínimo de tempo no mesmo local.

Em uma aplicação de turismo, um *stop* pode ser um ponto turístico, um hotel ou um aeroporto. Em um aplicativo de gerenciamento de tráfego, os lugares importantes (*stops*) podem ser semáforos, rótulas e parques de estacionamento. Dependendo da aplicação e das características dos dados, a duração mínima dessas paradas relevantes pode variar significativamente.

Definição 2: *Candidato a stop*. Um candidato a stop é definido como uma tupla (Rc e Δc) definida pelo usuário, onde Rc é a geometria do objeto geográfico de interesse e Δc é a duração mínima.

Em uma aplicação de turismo, por exemplo, a geometria de um hotel pode ser um candidato e a duração mínima de 3 horas seria o tempo em que uma trajetória deveria interceptar este hotel. Dependendo do candidato a stop, o tempo mínimo pode variar.

O conjunto de candidatos a stop caracteriza os aspectos importantes de uma aplicação.

Definição 3: *Aplicação.* Uma aplicação é um conjunto finito de candidatos a stop $A = \{C_N = (R_{cN} , \Delta_{cN})\}$ onde a geometria de cada candidato a stop não intercepta a geometria do outro. Essa limitação ocorre devido a restrições definidas pelo algoritmo CB_SMOT proposta por [Palma 2008].

Definição 4: *Stop*: Um stop é uma parte da trajetória que representa uma característica importante de um domínio de aplicação e precisa durar um tempo mínimo, podendo este interceptar ou não um candidato a stop.

Diversos métodos já foram citados para gerar stops. Em [Álvares 2007], por exemplo, um stop deve interceptar um candidato a stop por um tempo mínimo. Em [Palma 2008], um stop é uma região de velocidade baixa de uma trajetória por um mínimo período de tempo, podendo esta região interceptar ou não um candidato. Já em [Manso 2010] um stop é uma região da trajetória onde a direção teve uma variação significativa por um período de tempo. Em suma, a definição de stop varia com o método usado para gerá-lo.

Neste trabalho, um stop é definido como proposto por [Palma 2008].

Definição 5: *Move*: um move de uma trajetória T com relação ao conjunto de candidatos a stop é:

(v) a maior subtrajetória contínua de T entre dois stops consecutivos de T, ou;

(vi) a maior subtrajetória contínua de T entre o ponto inicial de T e o primeiro STOP de T; ou;

(vii) a maior subtrajetória contínua de T entre a última parada de T e o último ponto de T, ou;

(viii) a própria trajetória de T, se T não tem stops.

Em outras palavras, cada ponto de uma trajetória que não está no *stop* está em um *move*. Um *move* não tem tempo de duração mínimo e pode ou não cruzar com um candidato a *stop*. Caso isso aconteça, o intervalo de tempo de interseção deve ser menor do que o tempo mínimo do candidato a stop.

Palma [Palma2008] definiu conceitos para a mineração de trajetórias considerando o contexto de informações geográficas e também as propriedades geométricas das trajetórias, como a velocidade. Esses conceitos são apresentados na próxima seção.

2.3 O algoritmo CB-SMOT

O algoritmo CB-SMOT (Clustering-*Based Stops and Moves of Trajectories*), proposto por [Palma2008], tem duas etapas principais: clusterização e atribuição de semântica. Na parte de clusterização, CB-SMOT foi baseado em um algoritmo de densidade usando o conceito espaço-temporal. CB-SMOT considera adicionalmente o tempo no processo de clusterização, de forma que os clusters correspondem a trechos em que a velocidade é lenta.

No método CB-SMOT, as trajetórias são analisadas com base na sua geografia, no contexto que existe no espaço físico e na sua velocidade. Dessa forma, têm-se dois aspectos envolvidos com a trajetória: o primeiro é relacionado apenas com a trajetória (sua velocidade) e o segundo é o contexto espacial que existe por trás da trajetória (geografia). Cada aspecto é tratado de uma maneira independente. O primeiro utiliza clusterização para definir os trechos de menor velocidade. O segundo aspecto usa os candidatos a stop para atribuir semântica. Este aspecto relaciona a trajetória com um contexto que depende dos candidatos a stop de interesse do usuário.

O passo de clusterização, que identifica regiões de baixa velocidade, utiliza três parâmetros principais: *avg, speedLimit e minTime.* Inicialmente são encontradas as regiões de velocidade média abaixo do parâmetro *avg*, que representa a porcentagem média da velocidade. Esta etapa encontra todas as subtrajetórias com velocidade abaixo da velocidade média da trajetória inteira. Posteriormente essas regiões são expandidas, tentando-se agregar outros pontos de baixa velocidade. Dessa forma, vão sendo adicionados os pontos vizinhos que tenham velocidade inferior ao parâmetro velocidade máxima da trajetória (*SpeedLimit*) e que mantenham a velocidade média total da região sendo analisada, abaixo da velocidade média (*avg*). Terminada a

expansão dessas regiões, o próximo passo é escolher as regiões com menor velocidade para se tornarem um stop potencial. Se este stop potencial tiver a duração maior que o parâmetro mínimo de tempo *minTime*, ele é considerado um cluster e um stop é gerado.

O segundo passo (atribuição de semântica geográfica) diz respeito ao contexto geográfico da trajetória. Nesse momento, os trechos de interesse provenientes do passo anterior (stops/clusters com baixa velocidade) são verificados se interceptam os candidatos a stop. Dessa forma, podem-se ter dois tipos de stops: *known stops* e *unknown stops*. Os primeiros estão relacionados a pontos de interesse do usuário que existem na base de dados de candidatos a stops. Os unknown stops, por outro lado, são pontos que foram capturados no primeiro passo como clusters (devido a sua baixa velocidade), mas que pela seleção de interesse do usuário ou falta de candidatos a stops na base de dados, não intercepta nenhum candidato a stop, sendo por isso chamado de *Unknown stop*.

A Figura 3 ilustra um exemplo do resultado do algoritmo onde há a descoberta de *unkown* e *kown* stops.

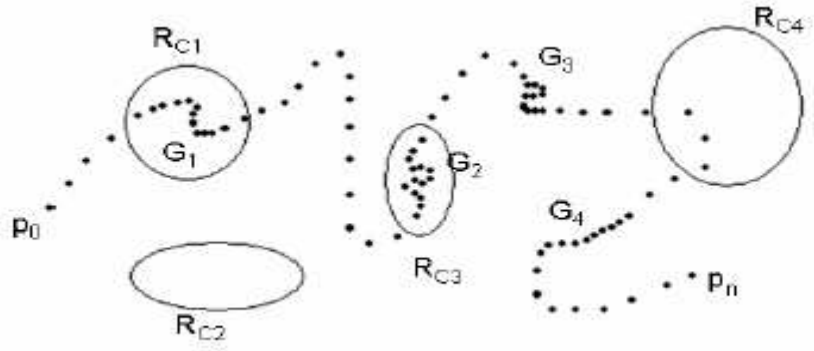


Figura 3– Exemplo de trajetória com 2 stops e 2 unknown stops (figura obtida em [Palma 2008]).

A trajetória da Figura 3 tem 4 stops possíveis, os clusters G_1, G_2, G_3 e G_4. Neste exemplo, o usuário informou 4 candidatos a stops, identificados pelas reticências R_{C1}, R_{C2}, R_{C3} e R_{C4}. O cluster G_1 intercepta o candidato a stop R_{C1} por um tempo superior a $_{C1}$. Então, o primeiro stop da trajetória é R_{C1}. O mesmo ocorre com o cluster G_2, considerando R_{C3}, que é o segundo stop da trajetória. Os clusters, G_3 e G_4 não interceptam qualquer candidato a stop. Portanto, G_3 e G_4 são stops desconhecidos. Cada stop desconhecido recebe uma identificação. No caso de dois ou mais stops desconhecidos se cruzarem, eles vão receber a mesma identificação (por exemplo, 4_unknown).

3. MÉTODOS PARA REDUÇÃO DE DADOS

Como o método CB-SMOT é baseado na variação da velocidade das trajetórias, ele é bastante interessante para aplicações de trânsito e por isso foi escolhido como o principal algoritmo do objeto de estudo deste trabalho. Trajetórias geradas por GPS em carros normalmente produzem um grande volume de dados. Considerando a demora de execução do algoritmo, verificou-se a necessidade de reduzir desse tempo de processamento. Assim sugiram duas opções: otimizar o algoritmo ou reduzir o volume de dados. Ao reduzir o volume de dados, outros algoritmos automaticamente serão beneficiados. Optou-se então por reduzir a quantidade de dados utilizando algumas técnicas de amostragens estatísticas como principais formas de otimização de tempo de processamento.

Uma das preocupações deste trabalho é que as técnicas de redução aplicadas garantam que a trajetória resultante da remoção e limpeza de dados represente semanticamente a trajetória original. Para isso, aplicaram-se algumas técnicas de amostragem usando alguns conceitos estatísticos de população e amostra.

Barbeta [Barbeta 2001] definiu população como sendo um conjunto de elementos passíveis de serem mensurados, com respeito às variáveis que se pretende levantar, podendo ser formada por pessoas, famílias, estabelecimentos industriais, ou qualquer outro tipo de elementos, dependendo basicamente dos objetivos da pesquisa. Considerando que o objeto de pesquisa deste trabalho é principalmente as trajetórias e seus respectivos stops, então população pode ser considerada como uma trajetória.

O termo amostragem [Lavado 2001] refere-se ao processo pelo qual se obtém uma amostra e deve ser realizada com técnicas adequadas para garantir a representatividade da população em estudo. Assim, essas técnicas têm que garantir que as trajetórias limpas representem a trajetória original. Ele ainda afirma que, sempre que possível, cada elemento da população deve ter igual chance de participar da amostra, evitando assim, o chamado viés de seleção. Isto significa que cada ponto da trajetória original tem que ter a mesma probabilidade de participar da trajetória limpa sem haver qualquer tipo de tendência ou influência.

[Barbeta 2001] afirma que técnicas de amostragem extraem do todo (população) uma parte (amostra) com o propósito de avaliar (inferir) algumas características da população, ou seja, a partir de técnicas de amostragem retira-se amostras que representem as características da população. Justificativas para o uso de amostragens:

4) *Economia.* Torna-se mais econômico o levantamento de somente uma parte da população.
5) *Tempo*. Para uma população consideravelmente grande, não há tempo viável de avaliar toda a população.
6) *Operacionalidade.* É mais fácil realizar operações de pequena escala.

Todas as três razões se justificam neste trabalho, uma vez que, ao se selecionar menos dados, o algoritmo executa mais rapidamente e como o volume de dados (população) também é grande, se torna inviável a execução do algoritmo devido ao tempo de processamento.

As formas de limpeza das trajetórias adotadas neste trabalho são compostas por dois métodos de amostragem e um de eliminação de ruídos, os quais são discutidos a seguir.

3.1. Eliminação de ruído

Ruído é considerado algum valor de velocidade ou tempo que esteja variando acima do parâmetro informado pelo usuário.

O usuário informa a velocidade ou o tempo máximo entre dois pontos da trajetória a ser limpa. O sistema calcula a velocidade e o tempo de todos os pontos da trajetória original. O algoritmo de limpeza verifica se a velocidade ou o tempo de cada ponto da trajetória original é maior que o parâmetro informado. Caso seja, então o ponto avaliado é removido da trajetória original. Caso contrário, o ponto fará parte da trajetória limpa.

Esta técnica de redução de dados muito se assemelha à forma como Pelekis [Pelekis 2008] utiliza os parâmetros gapTime e VMax na seção 3.1, que determinam se o ponto que está sendo avaliado do conjunto de pontos brutos vai fazer parte ou não da trajetória comparada. O primeiro parâmetro determina o máximo permitido de intervalo de tempo entre dois pontos, que pode ser comparado ao parâmetro tempo máximo proposto. Já o segundo determina a velocidade máxima permitida de um ponto na

trajetória, que tem o mesmo sentido do parâmetro velocidade máxima proposto nesta técnica. A diferença entre a técnica proposta e a técnica de Pelekis é que, na técnica de Pelekis, os pontos que são considerados como ruídos não são removidos inicialmente, uma vez que, estes pontos podem ser interessantes para outras trajetórias futuras. No entanto, as trajetórias finais reconstruídas não possuem os pontos ruidosos.

3.2 Amostragem Aleatória Simples

A amostra aleatória simples tem a seguinte propriedade [Barbeta 2001]: *qualquer subconjunto da população, com o mesmo número de elementos, tem a mesma probabilidade de fazer parte da amostra*. Em suma, tem-se que cada ponto da trajetória tem a mesma probabilidade de pertencer à amostra (trajetória limpa).

O usuário informa a porcentagem da trajetória original que vai fazer parte da trajetória limpa, ou seja, ele informa o tamanho da amostra, que não deixa de ser, o tamanho da trajetória limpa. Por exemplo, se para o usuário só interessar 50% dos dados da trajetória original, então ele informa ao sistema esta quantia e este sorteia 50% dos dados da trajetória original para a trajetória limpa e salva em uma nova tabela no banco de dados. Assim, temos uma nova tabela com menos pontos nas trajetórias.

3.3. Amostragem sistemática

Em [Unama 2010], os membros da população (pontos da trajetória) que participam da amostra são determinados a partir de intervalos fixos, e pode-se somente selecionar aleatoriamente o primeiro atributo (ponto). Por exemplo, uma amostra de pacientes internados em uma clínica. Sorteia-se um número de 1 a 5. Se o número sorteado for 2, incluem-se na amostra o paciente 2, o 7, o 12 e assim por diante variando de cinco em cinco.

Nesta técnica o usuário informa quantos pontos quer retirar entre dois pontos da trajetória a ser limpa. O primeiro ponto a ser selecionado é sorteado aleatoriamente entre 1 e 5. Os posteriores serão selecionados conforme o parâmetro determinado pelo usuário, ou seja, se o quinto(5°) ponto da trajetória original foi sorteado aleatoriamente como o primeiro ponto da trajetória limpa e o usuário tenha informado 4 como valor do parâmetro, então a trajetória limpa será constituída pelos pontos (5, 9, 13, 17, ...) da trajetória original.

4. EXPERIMENTOS E ANÁLISE DOS RESULTADOS

O conjunto de dados utilizado para avaliar as técnicas foram trajetórias geradas por aparelhos de GPS em automóveis que transitaram pela cidade do Rio de Janeiro entre o período 24 de fevereiro de 2002 a 16 de agosto de 2007. Esses dados constituem uma tabela no banco de dados com 1.000.000 de registros e contendo 325 trajetórias diferentes e em média 3.076 pontos por trajetória.

Houve dificuldade em achar o limite do número ou do percentual de pontos ideal em que se pode remover da trajetória original sem alteração nos resultados. Devido a esse valor depender exclusivamente da redundância de dados de cada conjunto de trajetórias, foram fixados os mesmos parâmetros de redução de dados para todas as tabelas de trajetórias, com o objetivo de avaliar se a aplicação de técnicas de redução compromete a semântica dos stops originais. Para técnica de redução por amostragem sistemática da seção 3.3 foram usados os parâmetros 1, 2, 3 e 4. Já para técnica de amostragem aleatória simples foram usados 50%, 33%, 25% e 20% como parâmetros.

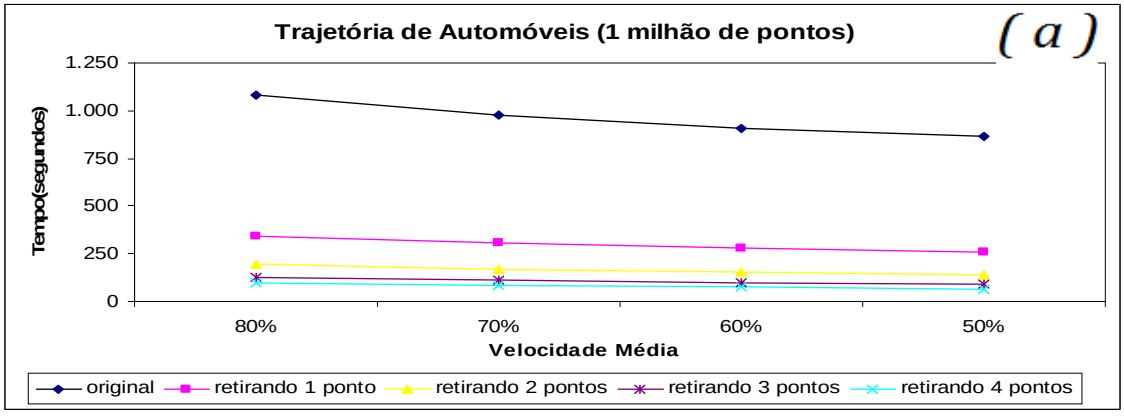
A seguir é avaliado quanto tempo de processamento se consegue otimizar com a aplicação das técnicas e posteriormente é avaliado os efeitos das redução de dados sobres os stops originais.

4.1. Análise do tempo de processamento

Para melhor avaliar o tempo de processamento do algoritmo CB-SMOT, foram fechados todos os programas adicionais do sistema operacional, salvo os realmente necessários para a execução do aplicativo Weka. Mesmo assim, os resultados podem ser divergentes, se executados em máquinas e sistemas operacionais diferentes. Em todos os casos, o algoritmo foi executado em um sistema operacional Windows Vista Home Basic, versão 1.6.0.20 do Java Development Kit(JDK), versão 8.4 do banco de dados Postgres, versão 1.4 do Postgis (complemento para consultas geográficas do Postgres) para uma máquina de 3 mega de memória RAM e processador Intel Core 2 Duo.

O Gráfico 1 representa o tempo de processamento do algoritmo CB-SMOT pela variação do parâmetro Velocidade Média (Average Speed) 0,5(50%) a 0,8(80%) da velocidade média de toda a trajetória. Os parâmetros fixos utilizados foram o Tempo Mínimo do stop (MinTime) com 600 segundos (10 minutos) e o Limite de Velocidade (Speed Limit) de 1,1. O Gráfico 1 utiliza uma tabela de trajetórias de automóveis que percorrem as vias do cidade do Rio de Janeiro. Para a tabela do gráfico 1a foi aplicada a técnica de amostragem sistemática, com os filtros 1, 2, 3 e 4 pontos, gerando assim quatro novas tabelas com respectivamente 500 mil, 330 mil, 250 mil e 200 mil pontos. Já para a tabela do gráfico 1b, foi aplicada a técnica de amostragem aleatória simples, citada no item 3.2, com os filtros 50%, 33%, 25% e 20%, gerando assim quatro novas tabelas com respectivamente 499.695, 333.174, 249.918 e 182.503 pontos.

A tabela original, que possui um milhão de pontos, é representada pela primeira e maior linha do gráfico. As demais linhas representam a tabela original com os dados reduzidos, podendo ser melhor conferidas na legenda do gráfico. No gráfico 1 pode ser observado que a exclusão de um único ponto, visto no gráfico 1a, ou a utilização de apenas 50% da trajetória original, visto no gráfico 1b, reduziu mais de 50% do tempo de processamento do algoritmo, independente da técnica de redução de dados.



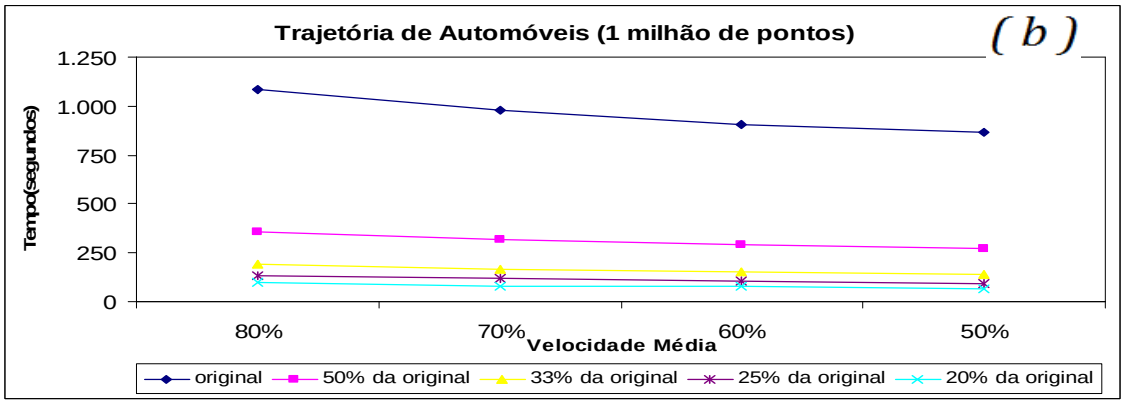


Gráfico 8 - Redução do tempo de processamento das trajetórias de carros.

4.1. Análise dos stops gerados

Nesta seção avaliaremos a qualidade do resultado do algoritmo após a limpeza dos dados.

Observando o Gráfico 2, é perceptível que, independente do tipo de limpeza utilizada, a quantidade de stops varia conforme aumenta a limpeza da trajetória original. Entretanto, pode-se observar que a variação no número de stops gerados após a limpeza é pequena para este conjunto de dados. Como mostra o Gráfico 2a, para uma velocidade média de 50%, a retirada de apenas um ponto diminuiu um stop, já a retirada de 2 ou 3 pontos aumentam 4 stops na trajetória limpa.

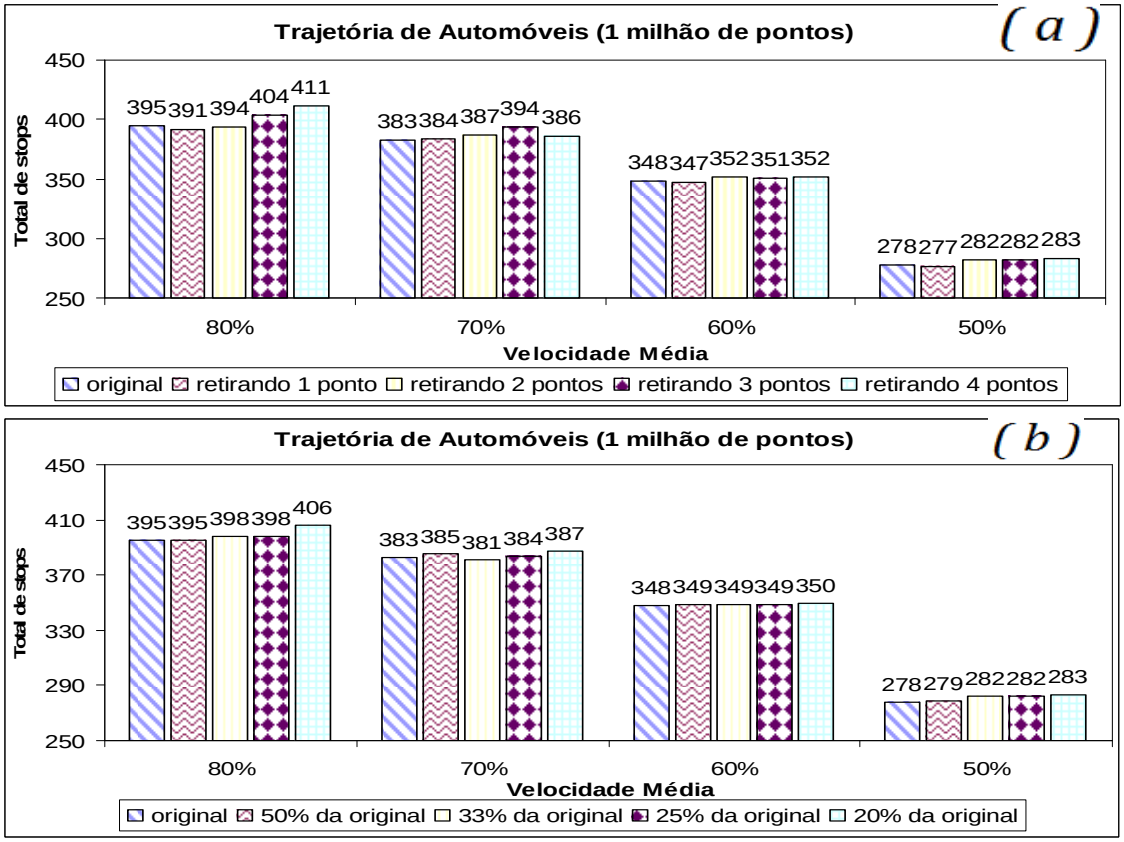


Gráfico 9 - Quantidade de stops por tabela.

Nos casos em que a trajetória limpa gerou menos stops que a trajetória original, estes stops sempre estão próximos ou até mesmo sobrepostos com os da trajetória original, podendo também ser de tamanho maior, considerando assim uma junção de

dois stops originais. A Figura 5 ilustra um exemplo, onde os dois stops inferiores da trajetória original foram aglutinados em apenas um stop nas trajetórias limpas. Note que os stops que se interceptam na trajetória original possuem o mesmo nome (*9_unknown*). Assim, observa-se que ambos os stops caracterizam uma mesma região lenta. Em seguida, observe que os stops das trajetórias limpas também possuem o mesmo nome (*9_unknown*). Com isso, entende-se que não houve perda de informação, a semântica dos stops é preservada, embora sua geometria tenha sido alterada.

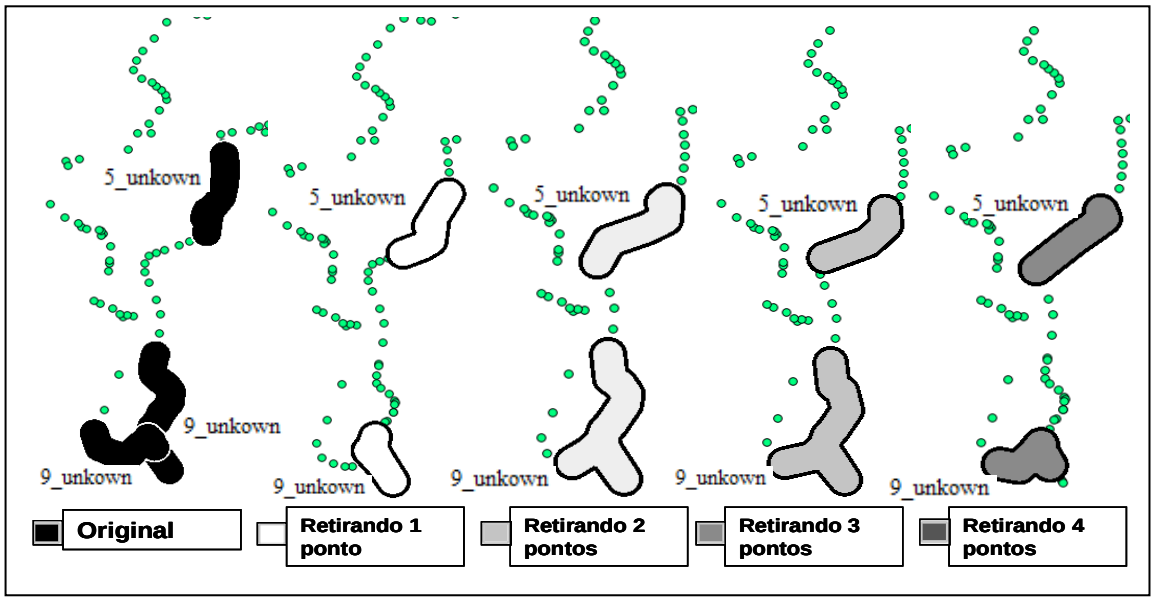


Figura 5 – Exemplo de limpeza e execução do algoritmo CB-SMOT onde os stops originais são em maior número que os stops das trajetórias limpas

Nos casos em que são gerados mais stops, considera-se que alguns pontos importantes do stop sejam excluídos na limpeza, e conseqüentemente quando os stops são calculados sobre as trajetórias limpas, o stop original se divide em dois ou três novos stops. A Figura 6 ilustra um stop original de tamanho grande que foi dividido a partir da retirada de dois pontos ou mais nas trajetórias limpas. Observe que, assim como na Figura 5, os nomes se mantiveram independente da aplicação das técnicas de limpeza. Embora tenham sido gerados mais stops após a limpeza dos dados, em todos os casos os stops permanecem como *0_unkown*. Com isso, entende-se que a semântica dos stops se mantém independente da retirada de um ou mais pontos.

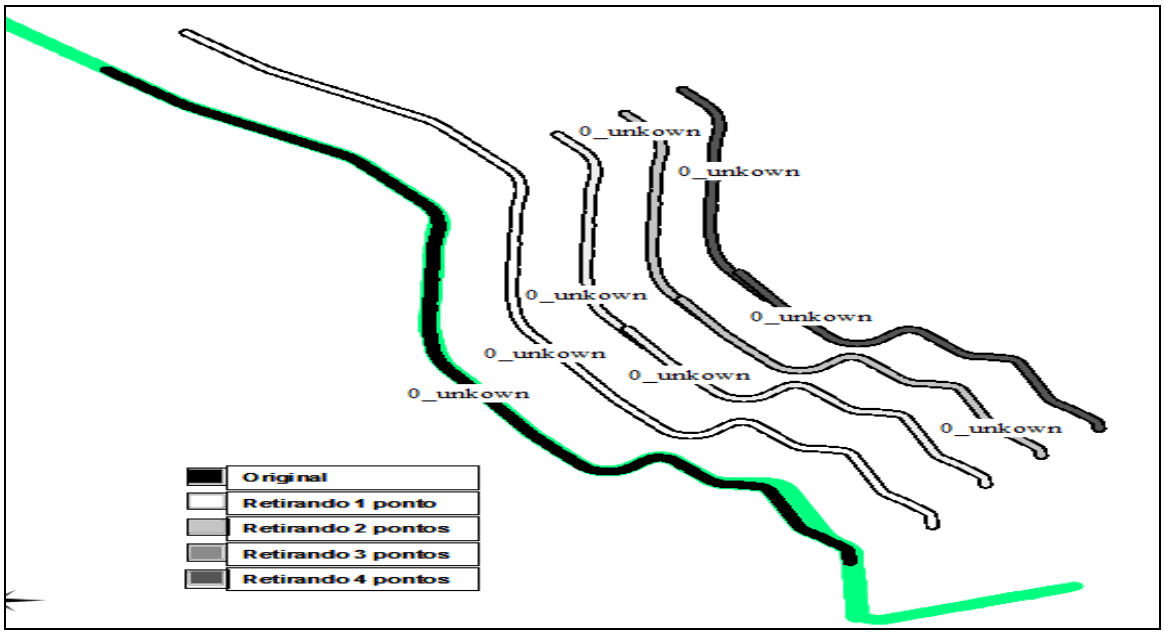


Figura 6 – Exemplo de limpeza e execução do algoritmo CB-SMOT na qual os stops originais foram divididos em dois os stops nas trajetórias limpas.

5. CONCLUSÃO E TRABALHOS FUTUROS

Com a popularização e acessibilidade de aparelhos de coleta de informações geográficas como aparelhos de GPS ou qualquer outro dispositivo móvel, aumentou significativamente disponibilidade de bases de dados espaço-temporais. Essas bases de dados são formadas por conjuntos de trajetórias de objetos móveis. Essas trajetórias são chamadas de trajetória brutas, pois não possuem contexto ou semântica associada. Com isso, foram criados alguns métodos de análise e extração de semântica sobre essas trajetórias brutas, sendo essa nova trajetória chamada trajetória semântica. No entanto, os tempos de processamento desses métodos são muito altos, podendo chegar a mais de 6 horas dependendo do volume de dados. Para isso, este trabalho vem com o objetivo de otimização deste tempo de processamento.

Como forma de otimização desse tempo de processamento, optou-se pela aplicação de técnicas de amostragem estatística sobre as trajetórias brutas antes da aplicação do algoritmo, com isso, tem-se que as trajetórias brutas ficam menores e mais limpas. Assim, gerando a hipótese de que se a quantidade de dados a ser analisada pelo algoritmo for menor, conseqüentemente, o tempo de processamento vai ser menor. Para isso, foram realizados experimentos sobre dois tipos de trajetórias diferentes: trajetórias de carros e de pedestres.

O objetivo geral das técnicas de redução de dados foram satisfatórias em relação ao tempo de processamento tanto para as trajetórias de carros como de pedestres, validando a hipótese anteriormente citada. Notou-se que, nos experimentos, houve uma otimização do tempo de processamento de no mínimo 55% utilizando a técnica de amostragem sistemática retirando apenas um ponto entre cada 2 pontos da trajetória original. Esta porcentagem se mantém igual para a técnica de amostragem aleatória simples com a utilização de 50% dos pontos da trajetória original.

Apesar de o tempo de processamento ter sido positivo independente da retirada de dados e dos tipos de trajetórias, os resultados do algoritmo foram diferentes do esperado. Para as trajetórias de carros a aplicação das técnicas de retirada de dados acarreta uma pequena diferença no número de stops e seu formato, mas essas diferenças não afetam a

semântica dos stops resultantes comparados com os stops originais. Neste sentido, o resultado foi satisfatório.

Este trabalho também identificou uma série de erros e problemas na implementação do algoritmo CB_SMOT que atrasaram a conclusão deste trabalho. A versão desse algoritmo utilizada para os experimentos foi com esse problema solucionado, porém fez com fossem reescritos todos os gráficos e análises dos stops.

Como trabalhos futuros, outras técnicas de redução de dados podem ser estudadas.

6. REFERÊNCIAS

[Alvares 2007] Alvares, L. O.; Bogorny, V.; Kuijpers, B.; Macedo, J. A. F.; Moelans, B.; Vaisman, A.: A Model for Enriching Trajectories with Semantic Geographical Information.. In: Proc. of the ACM 15th International Symposium on Advances in Geographic Information Systems (ACM-GIS'07), Seattle, Washingthon, 7-9 November (2007).pp. 162-169

[Álvares 2010] Alvares, Luis O. ; Palma, Andrey ; Oliveira, G. ; Bogorny, V. Weka-STPM: from trajectory samples to semantic trajectories. In: Workshop de Sofware Livre, 2010, Porto Alegre. WSL, 2010.

[Barbeta 2001] Barbeta, Pedro Allberto: *Estatística Aplicada ás Ciências Sociais.* 4° Edição, Editora UFSC, Florianópolis, SC, 2001. pp. 41-58.

[Lavado 2001] Lavado EL, Castro AA. Projeto de pesquisa (Parte V – amostra). In: Castro AA. Planejamento da Pesquisa. São Paulo: AAC; 2001. Disponível em URL: Projeto de pesquisa; Acessado em agosto de 2010.

[Manso 2010] Manso, J. A; Times, V. C.; Oliveira, G.; Alvares, L.O.; Bogorny, V. DB-SMoT: a Direction-based spatio-temporal clustering method. Fifth IEEE International Conference on Intelligent Systems (IEEE IS 2010), 2010.

[Palma 2008] Palma, A. T; Bogorny, V.; Kuijpers, B.; Alvares, L.O. A Clustering-based Approach for Discovering Interesting Places in Trajectories. In: 23rd Annual Symposium on Applied Computing, (ACM-SAC'08), Fortaleza, Ceara, 16-20 March (2008) Brazil. pp. 863-868.

[Pelekis 2008] Pelekis, N.; Marketos, G.; Frentzos, E.; Ntoutsi, I.; Raffaetà, A.; Theodoridis, T.; "Building Real-World Trajectory Warehouses" In: 7th International ACM Workshop on Data Engineering for Wireless and Mobile Access, Vancouver, Canada, 2008, pp. 8–15.

[Spaccapietra 2008] Spaccapietra, S.; Parent, C.; Damiani, M.L.; Macedo, J. A. F. de ; Porto, F.; Vangenot, C.; A conceptual view on trajectories, Data & Kownledge Egineering, 65 n.1, p126-146, April 2008.

Anexo 2 – Código Fonte.

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package weka.gui.stpm.clean;

import java.io.IOException;
import java.sql.Statement;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Date;
import java.util.List;
import java.util.Vector;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JOptionPane;
import weka.gui.stpm.Config;
import weka.gui.stpm.GPSPoint;
import weka.gui.stpm.Trajectory;
import weka.gui.stpm.TrajectoryFrame;

/**
 *
 * @author hercules
 */
public class TrajectoryClean {

    private static FilterClean filterClean = FilterClean.getInstance();

    public static boolean isPrintSpeedToFileXls() {
        return false;
    }
    private Connection conn;
    /**User configurations*/
    private Config config = new Config();

    private String esquema;

    public static FilterClean getFilterClean() {
        return filterClean;
    }
    private TrajectoryFiltersCleanFrame trajectoryFiltersCleanFrame;

    public TrajectoryClean(Connection conn, List<String> listTabela) {
        this.conn = conn;
        this.filterClean.setListNomeTabelas(listTabela);
        this.trajectoryFiltersCleanFrame = new TrajectoryFiltersCleanFrame(this);
    }
    public TrajectoryClean(Connection conn) {
        this.conn = conn;
        this.trajectoryFiltersCleanFrame = new TrajectoryFiltersCleanFrame(this);
    }

    public String getEsquema() {
        if(this.esquema ==null){
            return "";
```

```
        }
        return esquema;
    }

    public void setEsquema(String esquema) {
        this.esquema = esquema;
    }

    public void filtrarTrajetoria() {

        createCleanTrajCreate();
       for (String nomeTabela : this.filterClean.getListNomeTabelas()) {
            java.util.Date tempo, fim, ini = new java.util.Date();
            try {
                String[] nmAtts = criarNovaTabelaTrajetoria(nomeTabela);
                getTrajectoriesWithSpeeds(nomeTabela, this.config, null, nmAtts[0], nmAtts[1]);

                fim = new java.util.Date();
                tempo = new java.util.Date(fim.getTime() - ini.getTime());
                inserirCleanTrajCreate(TrajectoryClean.getNomeTabNova(nomeTabela), tempo, nomeTabela);
                System.out.println("Processing time: " + tempo.getTime() + " ms");
                JOptionPane.showMessageDialog(this.trajectoryFiltersCleanFrame, "Operation finished
succesfully.");
            } catch (Exception e) {
                e.printStackTrace();
                JOptionPane.showMessageDialog(this.trajectoryFiltersCleanFrame, "Error during operation");
                System.out.println("Error: \n" + e.getMessage());
            }

        }//endof foreach
        Runtime.getRuntime().gc();
    }

    private void createCleanTrajCreate() {
        try {
            ResultSet rsn = executeQuery("select relname from pg_class where
lower(relname)='cleantrajcreate'");
            if (!rsn.next()) {
                StringBuilder b = new StringBuilder();
                b.append(" CREATE TABLE cleanTrajCreate ");
                b.append(" ( ");
                b.append("   id serial NOT NULL, ");
                b.append("   nomeTrajetoria character varying, ");
                b.append("   oidTrajetoria oid, ");
                b.append("   tempoLimpeza integer, ");
                b.append("   paramVeloci double precision, ");
                b.append("   paramTempo double precision, ");
                b.append("   paramQtePontos integer, ");
                b.append("   paramPorcent double precision ");
                b.append(" )  ");
                execute(b.toString());
                System.out.println("Creating tables createCleanTrajCreate...");
            }

            try {
                rsn.close();
            } catch (SQLException ex) {
                Logger.getLogger(TrajectoryClean.class.getName()).log(Level.SEVERE, null, ex);
            }
        } catch (Exception e) {
```

```
            e.printStackTrace();
        }finally{
        }
    }

    private List<TrajetoryCleanDto> pegarDadosTrajetoria(String nomeTabela) {
        List<TrajetoryCleanDto> lista = new ArrayList<TrajetoryCleanDto>();
        try {
            StringBuilder b = new StringBuilder();
            b.append(" CREATE TABLE temporaryTrajClean as select gid, time, tid, the_geom from ").append(nomeTabela).append(" order by tid, time, gid; ");
            b.append(" alter table temporaryTrajClean add linha serial;");
            execute(b.toString());

            b = new StringBuilder();
            b.append(" select gid, tid, tempo, tempo2, (distancia/tempo) as velocidade from ( ");
            b.append(" select t.linha, t.gid, t.tid,  ");
            b.append(" (        st_distance(t0.the_geom,t1.the_geom) + ");
            b.append("          st_distance(t1.the_geom,t2.the_geom) + ");
            b.append("          st_distance(t2.the_geom,t3.the_geom) + ");
            b.append("          st_distance(t3.the_geom,t4.the_geom) ");
            b.append(" )as distancia, ");
            b.append("  ");
            b.append(" ( ");
            b.append("          extract(EPOCH from (to_timestamp(t0.time, 'YYYY-MM-DD HH24:MI:SS')-to_timestamp(t1.time, 'YYYY-MM-DD HH24:MI:SS'))) + ");
            b.append("          extract(EPOCH from (to_timestamp(t1.time, 'YYYY-MM-DD HH24:MI:SS')-to_timestamp(t2.time, 'YYYY-MM-DD HH24:MI:SS'))) + ");
            b.append("          extract(EPOCH from (to_timestamp(t2.time, 'YYYY-MM-DD HH24:MI:SS')-to_timestamp(t3.time, 'YYYY-MM-DD HH24:MI:SS'))) + ");
            b.append("          extract(EPOCH from (to_timestamp(t3.time, 'YYYY-MM-DD HH24:MI:SS')-to_timestamp(t4.time, 'YYYY-MM-DD HH24:MI:SS'))) ");
            b.append(" )as tempo, ");
            b.append("  ");
            b.append(" extract(EPOCH from (to_timestamp(t2.time, 'YYYY-MM-DD HH24:MI:SS')-to_timestamp(t3.time, 'YYYY-MM-DD HH24:MI:SS'))) as tempo2 ");
            b.append(" from temporaryTrajClean t  ");
            b.append(" left join temporaryTrajClean t0 on t.linha=t0.linha-2 and t.tid=t0.tid ");
            b.append(" left join temporaryTrajClean t1 on t.linha=t1.linha-1 and t.tid=t1.tid ");
            b.append(" left join temporaryTrajClean t2 on t.linha=t2.linha   and t.tid=t2.tid ");
            b.append(" left join temporaryTrajClean t3 on t.linha=t3.linha+1 and t.tid=t3.tid ");
            b.append(" left join temporaryTrajClean t4 on t.linha=t4.linha+2 and t.tid=t4.tid ");
            b.append(" order by linha ");
            b.append(" )as foo; ");

            ResultSet rsn = executeQuery(b.toString(), "Erro na consulta da velocidade da trajetoria.");

            while (rsn.next()) {
                TrajetoryCleanDto dto = new TrajetoryCleanDto();
                dto.setGid(rsn.getLong("gid"));
                dto.setTid(rsn.getInt("tid"));
                dto.setTempo2(rsn.getDouble("tempo2"));
                dto.setTempo(rsn.getDouble("tempo"));
                dto.setVelocidade(rsn.getDouble("velocidade"));
                lista.add(dto);
            }

            b=new StringBuilder();
            b.append(" DROP TABLE temporaryTrajClean ");
            execute(b.toString());
```

```
//          deletarSequenceLinha(nomeTabela);
      } catch (Exception e) {
         e.printStackTrace();
      }
      return lista;
   }
   public void getTrajectoriesWithSpeeds(String tableTraj, Config config, Integer table_srid,String nmAttSelect,String nmAttGroup) throws SQLException, IOException {

      Statement s = config.conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
      // selects the trajectory-tid to be processed
      String sql = "SELECT "+config.tid+" as tid, count(*) FROM "+tableTraj+" GROUP BY "+config.tid+" ORDER BY tid DESC";
      ResultSet rs = s.executeQuery(sql);

         //for each trajectory...
      while (rs.next()) {
         Trajectory trajectory = null;
         if(table_srid != null){
            trajectory = new Trajectory(table_srid);
         }else{
            trajectory = new Trajectory();
         }
         trajectory.tid = rs.getInt("tid");

         //select the points of the trajectory in sequential time
         Statement s1 = config.conn.createStatement();
         String sql2 = "SELECT "+config.time+" as time,the_geom,gid FROM "+tableTraj+" WHERE "+config.tid+"="+trajectory.tid+" ORDER BY "+config.time;
         //System.out.println(sql2);

         ResultSet rs2 = s1.executeQuery(sql2);
         //for each of these points of the trajectory...
         int timeIndex = 0;
         while (rs2.next()) {
            Timestamp t = rs2.getTimestamp("time");
            org.postgis.PGgeometry geom = (org.postgis.PGgeometry) rs2.getObject("the_geom");
            org.postgis.Point p = (org.postgis.Point) geom.getGeometry();
            //get the time, the_geom and tid columns to fill the Vector
            GPSPoint gps = new GPSPoint(trajectory.tid,t,p,timeIndex);
            gps.gid = rs2.getInt("gid");
            trajectory.points.addElement(gps);
            timeIndex++;

         }
         System.out.println("calculando velocidade da trajetoria "+trajectory.tid);
         retirarDados3(trajectory, tableTraj, nmAttSelect, nmAttGroup  );
         System.out.println("add trajetoria " + trajectory.tid);
         System.gc();
      }
      return ;
   }
   private List<Trajectory> pegarDadosTrajetoria2(String nomeTabela) {
      List<Trajectory> lista = new ArrayList<Trajectory>();
      try {
         return TrajectoryFrame.getTrajectoriesWithSpeeds(nomeTabela, this.config, null);
      } catch (Exception e) {
         e.printStackTrace();
```

```
        }
        return lista;
    }
    private void retirarDados3(Trajectory traj, String nomeTabela, String nmAttSelect, String nmAttGroup)
{
        StringBuilder ids = new StringBuilder("   ");

            Vector<GPSPoint> pontos = traj.points;

            if (TrajectoryClean.getFilterClean().getPorcentagem() != null) {
                Collections.shuffle(pontos);
                int probFim = (int)(pontos.size() * TrajectoryClean.getFilterClean().getPorcentagem() / 100);
                List<GPSPoint> listTraj0 = pontos.subList(0, probFim);
                for (GPSPoint pt : listTraj0) {
                    ids.append(pt.gid).append(", ");
                }

            } else if (TrajectoryClean.getFilterClean().getQtdePontos() != null) {
                if (TrajectoryClean.getFilterClean().getVelocidadeMaxima() != null) {
                    if (traj.points.size()>5){
                        traj.calculatePointsSpeed();
                    }
                }
                //retorna Math.round(Math.random()*5) --- de 0 a 5, incluindo o 5;
                int i = (int) Math.round(Math.random() * 5);
                while (i < traj.points.size()) {
                    GPSPoint pt = traj.points.get(i);
                    i      +=      TrajectoryClean.getFilterClean().getQtdePontos()      ==      null      ?      1      :
(this.getFilterClean().getQtdePontos()+1);
                    if (TrajectoryClean.getFilterClean().getVelocidadeMaxima() != null) {
                        if (pt.speed >= TrajectoryClean.getFilterClean().getVelocidadeMaxima()) {
                            continue;
                        }
                    }
                    if (TrajectoryClean.getFilterClean().getTempoMaximo() != null) {
                    }
                    ids.append(pt.gid).append(", ");
                }
            }
            if(ids.toString().trim().length()>0){
                inserirNovaTabelaTrajetoria(nomeTabela,                                ids.toString().trim().substring(0,
ids.toString().trim().length() - 1), nmAttSelect, nmAttGroup);
                System.out.println("Inseriu estes ids = "+ids.toString());
            }
            ids = new StringBuilder("   ");
            System.gc();
    }
    private String retirarDados2(List<Trajectory> listTraj, String nomeTabela) {
        String[] nmAtts = criarNovaTabelaTrajetoria(nomeTabela);
        StringBuilder ids = new StringBuilder("   ");

        for (Trajectory dto : listTraj) {
            Vector<GPSPoint> pontos = dto.points;

            if (TrajectoryClean.getFilterClean().getPorcentagem() != null) {
                Collections.shuffle(pontos);
                int probFim = (int)(pontos.size() * TrajectoryClean.getFilterClean().getPorcentagem() / 100);
                List<GPSPoint> listTraj0 = pontos.subList(0, probFim);
                for (GPSPoint pt : listTraj0) {
                    ids.append(pt.gid).append(", ");
```

```
            }

        } else if (this.getFilterClean().getQtdePontos() != null) {
            this.getFilterClean().getQtdePontos();

            //retorna Math.round(Math.random()*5) --- de 0 a 5, incluindo o 5;
            int i = (int) Math.round(Math.random() * 5);
            while (i < dto.points.size()) {
                GPSPoint pt = dto.points.get(i);
                i += this.getFilterClean().getQtdePontos() == null ? 1 :
(this.getFilterClean().getQtdePontos()+1);
                if (this.getFilterClean().getVelocidadeMaxima() != null) {
                    if (pt.speed >= this.getFilterClean().getVelocidadeMaxima()) {
                        continue;
                    }
                }
                if (this.getFilterClean().getTempoMaximo() != null) {
                }
                ids.append(pt.gid).append(", ");
            }
        }
        inserirNovaTabelaTrajetoria(nomeTabela, ids.toString().trim().substring(0,
ids.toString().trim().length() - 1), nmAtts[0], nmAtts[1]);
        System.out.println("Inseriu estes ids = "+ids.toString());
        ids = new StringBuilder("   ");
        System.gc();
      }
      return ids.toString().substring(0, ids.length() - 3);
   }
   private String retirarDados(List<TrajetoryCleanDto> listTraj) {

      StringBuilder b = new StringBuilder("   ");

      if (TrajectoryClean.getFilterClean().getPorcentagem() != null) {
         Collections.shuffle(listTraj);
         int probFim = (int)(listTraj.size() * TrajectoryClean.getFilterClean().getPorcentagem() / 100);
         List<TrajetoryCleanDto> listTraj0 = listTraj.subList(0, probFim);
         for (TrajetoryCleanDto dto : listTraj0) {
            b.append(dto.getGid()).append(", ");
         }

      } else if (this.getFilterClean().getQtdePontos() != null) {
         this.getFilterClean().getQtdePontos();

         //retorna Math.round(Math.random()*5) --- de 0 a 5, incluindo o 5;
         int i = (int) Math.round(Math.random() * 5);
         while (i < listTraj.size()) {
            TrajetoryCleanDto traj = listTraj.get(i);
            i += this.getFilterClean().getQtdePontos() == null ? 1 :
(this.getFilterClean().getQtdePontos()+1);
            if (this.getFilterClean().getVelocidadeMaxima() != null) {
               if (traj.getVelocidade() >= this.getFilterClean().getVelocidadeMaxima()) {
                  continue;
               }
            }
            if (this.getFilterClean().getTempoMaximo() != null) {
               if (traj.getTempo() >= this.getFilterClean().getTempoMaximo()) {
                  continue;
               }
            }
```

```
            b.append(traj.getGid()).append(", ");
        }
      }
      return b.toString().substring(0, b.length() - 3);
   }

   public static String getNomeTabNova(String nomeTabela) {
      if(getFilterClean().toString()!=null && !getFilterClean().toString().equals("")){
         return nomeTabela.concat("_").concat(getFilterClean().toString());
      }
      return nomeTabela;
   }

    private String[] criarNovaTabelaTrajetoria(String nomeTabela) {
      StringBuilder nmAttrsFrom    = new StringBuilder();
      StringBuilder nmAttrGroup = new StringBuilder(" tid, time ");

       //pegar os atributos da tabela attual
      try {

         StringBuilder strSelect = new StringBuilder();
         StringBuilder strInsere = new StringBuilder();
         strSelect.append(" select Att.attname,Type.typname, Att.attnotnull from pg_attribute Att ");
         strSelect.append(" JOIN pg_class Tab ON (Att.attrelid = Tab.oid)  ");
         strSelect.append(" JOIN pg_type Type ON (Att.atttypid = Type.oid)  ");
         strSelect.append(" WHERE Att.attnum > 0  ");
         strSelect.append("      and      Type.typname      <>      'geometry'      AND      Tab.relname      =
'").append(nomeTabela).append("' ");
         strSelect.append(" order by Att.attnum ");
         ResultSet rsAttr = executeQuery(strSelect.toString());

         strInsere.append(" DROP TABLE ").append(this.getNomeTabNova(nomeTabela)).append("; ");
         execute(strInsere.toString());
         strInsere = new StringBuilder();
         strInsere.append(" CREATE TABLE ").append(getNomeTabNova(nomeTabela)).append(" (   gid
serial NOT NULL ");

         while (rsAttr.next()) {
            strInsere.append(", ");
            String nmAtt = rsAttr.getString("attname");
            nmAttrsFrom.append(nmAtt);
            if("gid".equalsIgnoreCase(nmAtt)){
               nmAttrsFrom.append(" as gidOld ");
               nmAtt="gidOld";
               nmAttrGroup.append(", ");
               nmAttrGroup.append(nmAtt);

            }else if(!"tid".equalsIgnoreCase(nmAtt) && !"time".equalsIgnoreCase(nmAtt)){
               nmAttrGroup.append(", ");
               nmAttrGroup.append(nmAtt);
            }
            strInsere.append(nmAtt).append(" ");
            strInsere.append(rsAttr.getString("typname")).append(" ");
            if (rsAttr.getBoolean("attnotnull")) {
               strInsere.append(" NOT NULL ");
            }
            if (!rsAttr.isLast()) {
                  nmAttrsFrom.append(", ");
            }
         }
```

```
        try {
           rsAttr.close();
        } catch (SQLException e) {
           e.printStackTrace();
        }
        strInsere.append(" ) ");
        execute(strInsere.toString());

        //retorna todos os nome dos atributos das chaves primarias.
        strSelect = new StringBuilder();
        strSelect.append(" select Att.attname from pg_attribute Att ");
        strSelect.append(" JOIN pg_class Tab ON (Att.attrelid = Tab.oid)  ");
        strSelect.append(" WHERE Att.attnum > 0  ");
        strSelect.append(" and array[Att.attnum] <@ ( select conkey from pg_constraint cons where
cons.conrelid=Tab.oid and cons.contype='p') ");
        strSelect.append(" AND Tab.relname = '").append(nomeTabela).append("' ");
        rsAttr = executeQuery(strSelect.toString());

        strInsere = new StringBuilder();
        strInsere.append(" ALTER TABLE ").append(this.getNomeTabNova(nomeTabela));
        strInsere.append(" ADD CONSTRAINT ").append(this.getNomeTabNova(nomeTabela) + "_pk");
        strInsere.append(" PRIMARY KEY ( ");
        while (rsAttr.next()) {
           strInsere.append(rsAttr.getString("attname"));
           if (!rsAttr.isLast()) {
              strInsere.append(", ");
           }
        }
        try {
           rsAttr.close();
        } catch (SQLException e) {
           e.printStackTrace();
        }
        strInsere.append(" ); COMMIT; ");

        execute(strInsere.toString());


        //pega todos os nome das coluna do tipo GEOMETRY
        strSelect = new StringBuilder();
        strSelect.append(" select Att.attname from pg_attribute Att ");
        strSelect.append(" JOIN pg_class Tab ON (Att.attrelid = Tab.oid)  ");
        strSelect.append(" JOIN pg_type Type ON (Att.atttypid = Type.oid)  ");
        strSelect.append(" WHERE Att.attnum > 0 and Type.typname = 'geometry' AND Tab.relname  =
'').append(nomeTabela).append("' ");
//          System.out.println(strSelect.toString());
//          rsAttr = stateSelect.executeQuery(strSelect.toString());
        rsAttr = executeQuery(strSelect.toString());

        // para cada coluna do tipo GEOMETRY na tab cria na nova
        while (rsAttr.next()) {
           String nmAttr = rsAttr.getString("attname");
           nmAttrsFrom.append(", ").append(nmAttr);
           nmAttrGroup.append(", ").append(nmAttr);
           String str = "select distinct st_srid(" +nmAttr
                 + ") as srid, st_ndims(" +nmAttr
                 + ") as ndims, geometrytype(" +nmAttr
                 + ") as geometrytype from " + nomeTabela;
//            System.out.println(str);
//            ResultSet rsAttrGeo = stateSelect2.executeQuery(str);
```

```
            ResultSet rsAttrGeo = executeQuery(str);
            strInsere = new StringBuilder();
            strInsere.append("SELECT AddGeometryColumn('"
                  + this.getNomeTabNova(nomeTabela) + "','"
                  +nmAttr + "','");

            if(rsAttrGeo.next()){
               strInsere.append(rsAttrGeo.getInt("srid") + "','"
                  + rsAttrGeo.getString("geometrytype") + "',"
                  + rsAttrGeo.getInt("ndims"));
            }else{
                strInsere.append(" '-1', 'POINT', 2 ");
            }
            strInsere.append(" ) ");
//              System.out.println(strInsere.toString());
//              stateInsert.execute(strInsere.toString());
            execute(strInsere.toString());
            try {
               rsAttrGeo.close();
            } catch (SQLException e) {
               e.printStackTrace();
            }
         }
         try {
            rsAttr.close();
         } catch (SQLException e) {
            e.printStackTrace();
         }



      } catch (SQLException ex) {
         ex.printStackTrace();
      }
      return new String[]{nmAttrsFrom.toString(), nmAttrGroup.toString()};
   }



   private void inserirNovaTabelaTrajetoria(String nomeTabela, String ids, String nmAttrsSelect, String
nmAttrsGroup) {
      //pegar os atributos da tabela attual
         StringBuilder strInsere = new StringBuilder();
         String nmTabNova = TrajectoryClean.getNomeTabNova(nomeTabela);
         strInsere.append("INSERT INTO ").append(nmTabNova);
         strInsere.append("              SELECT              nextval('").append(nmTabNova).append("_gid_seq'),
").append(nmAttrsSelect).append(" from ").append(nomeTabela);
         strInsere.append(" where gid in ( ").append(ids).append(" ) ");
         strInsere.append(" group by ").append(nmAttrsGroup);
         strInsere.append(" order by tid, time ");
         execute(strInsere.toString(), "Erro ao inserir dados na nova trajetória.");


   }

   private void inserirCleanTrajCreate(String nomeTabelaNova, Date tempoCriacaoNova, String
nomeTabelaVelha) {
      //pegar os atributos da tabela attual
//       try {
//          Statement stateInsert = this.conn.createStatement();
```

```
        StringBuilder strInsere = new StringBuilder();
        strInsere.append("INSERT INTO cleantrajcreate ");
        strInsere.append(" ( nomeTrajetoria, oidTrajetoria, tempoLimpeza, pontosAntes, pontosDepois ,
paramVeloci, paramTempo, paramQtePontos, paramPorcent )");
        strInsere.append(" VALUES ( '").append(nomeTabelaNova).append("', ");
        strInsere.append(" null ").append(", '");
        strInsere.append(tempoCriacaoNova.getTime()).append("', ");

        strInsere.append(" (select count(*) from ").append(nomeTabelaNova ).append(" ) , ");
        strInsere.append(" (select count(*) from ").append(nomeTabelaVelha).append(" ) , ");
        if(this.getFilterClean().getVelocidadeMaxima()==null){
           strInsere.append(" null ");
        }else{
           strInsere.append(this.getFilterClean().getVelocidadeMaxima());
        }
        strInsere.append(", ");
        if(this.getFilterClean().getTempoMaximo()==null){
           strInsere.append(" null ");
        }else{
           strInsere.append(this.getFilterClean().getTempoMaximo());
        }
        strInsere.append(", ");

        if(this.getFilterClean().getQtdePontos()==null){
           strInsere.append(" null ");
        }else{
           strInsere.append(this.getFilterClean().getQtdePontos());
        }
        strInsere.append(", ");

        if(this.getFilterClean().getPorcentagem()==null){
           strInsere.append(" null ");
        }else{
           strInsere.append(this.getFilterClean().getPorcentagem());
        }
        strInsere.append(" ) ");
        execute(strInsere.toString());
  }

   public void setVisibleFrame(boolean b) {
      this.trajectoryFiltersCleanFrame.setVisible(b);
   }

   private void execute(String sql, String msg) {
      Statement state = null;
       try {
          state = this.conn.createStatement();
          System.out.println(sql);
          state.execute(sql);
          state.close();
       } catch (SQLException ex) {
          Logger.getLogger(TrajectoryClean.class.getName()).log(Level.SEVERE, null, ex);
          if(msg != null){
             msg = "".equals(msg)?"Erro na consulta do banco de dados":msg;
             JOptionPane.showMessageDialog(this.trajectoryFiltersCleanFrame, msg);
          }
       }

   }
```

```
    private void execute(String sql) {
        String msg = null;
        this.execute(sql, msg);
    }



    private ResultSet executeQuery(String sql, String msg) {
        Statement state = null;
        ResultSet rsn = null;
        try {
            state = this.conn.createStatement();
            System.out.println(sql);
            rsn = state.executeQuery(sql);
//            state.close();
        } catch (SQLException ex) {
            Logger.getLogger(TrajectoryClean.class.getName()).log(Level.SEVERE, null, ex);
            if(msg != null){
                msg = "".equals(msg)?"Erro na consulta do banco de dados":msg;
                JOptionPane.showMessageDialog(this.trajectoryFiltersCleanFrame, msg);
            }
        }
        return rsn;
    }
    private ResultSet executeQuery(String sql) {
        String msg = null;
        return executeQuery(sql, msg);
    }

    private void deletarSequenceLinha(String nomeTabela) {

    }

    private void criarSequenceLinha(String nomeTabela) {

    }

    /**
     * @return the config
     */
    public Config getConfig() {
        return config;
    }

    /**
     * @param config the config to set
     */
    public void setConfig(Config config) {
        this.config = config;
    }
}
```

```
package weka.gui.stpm;

import javax.swing.*;

import weka.gui.WekaTaskMonitor;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;
import java.awt.event.*;
import java.io.File;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.List;
import java.util.Vector;
import java.util.logging.Level;
import java.util.logging.Logger;
import weka.gui.geodata.visualizer.ShowGeoData;
import weka.gui.stpm.clean.TrajectoryClean;
import weka.gui.stpm.clean.Util;


//public class TrajectoryFrame extends JFrame{
public class TrajectoryFrame extends JDialog{

        /**set of methods used */
        private Method[] algs;

        /**Connection with the DB*/
        private Connection conn;

        /**User configurations*/
        private Config config = new Config();

        // Method panel
        /**Selects the apropriate method to be aplicated */
    private javax.swing.JComboBox jComboBoxMethod;

    /**Select the parameters of each method */
    private javax.swing.JComboBox jComboBoxParam;

    /**Enters the numbers atributed to each parameter here. */
    private javax.swing.JTextField jTextFieldParam;

    /**The name of the table with streets information. */
    private javax.swing.JComboBox jComboBoxStreet;

    /**The column of that table with information about speed limit (in meters/second). */
```

```
private javax.swing.JComboBox jComboBoxStreetLimit;

//Relevant Features, Buffer and MinTime
/**List of relevant features, the tables to use to try to discover the places in trajectory.*/
private javax.swing.JList jListRF;

/**Indicates the distance in meters of the buffer in the relevant features. */
//private javax.swing.JSpinner jSpinnerBuffer;
private javax.swing.JTextField jTextFieldBuffer;

/**Use or not buffer in the relevant features */
private javax.swing.JCheckBox jCheckBoxBuffer;

/**Text field for entering the min time. Must enter the seconds and press 'TAB', for recording. */
        private javax.swing.JTextField RFMinTime;

//Feature type/instance
/**If selected, says that we are working with Feature Instance. */
private javax.swing.JRadioButton jRadioButtonFInstance;

/**If selected, says that we are working with Feature Type. */
private javax.swing.JRadioButton jRadioButtonFType;

//others...
/**Opens the Generate Arff File Frame */
private javax.swing.JButton jButtonGenArffFile;

/** trying to make the bird to fly... */
private final WekaTaskMonitor tm = new WekaTaskMonitor();

/**The thing (trajectories tables) which we want to aply the method selected*/
private javax.swing.JList jListTrajectoryTables; // replaces the jComboBoxTF

/**Select the schema of DB, usually 'public' for localhost tests. */
private javax.swing.JComboBox jComboBoxSchema;

/**DEPRECATED*/
private javax.swing.JComboBox jComboBoxTF; //old targetfeature combbox

private javax.swing.ButtonGroup buttonGroup1;//used in the genarfffile frame
private javax.swing.ButtonGroup buttonGroupItem;// the same
private javax.swing.ButtonGroup buttonGroupTime;//the same

/**Spatial reference for ALL trajectory_table*/
private int table_srid; //spatial reference ID,google it

/**Buffer variable*/
private Double buffer=50.0;       // variable buffer, initialized

public TrajectoryFrame(){
        this.setTitle("Trajectory");
   init();
        initComponents();
}

public TrajectoryFrame(String user, String pass, String url) {
        this.setTitle("Trajectory");
        init();
   initComponents();
        try {
```

```
            if (user == "")
               conn = DriverManager.getConnection(url);
            else
               conn = DriverManager.getConnection(url,user,pass);

            ((org.postgresql.PGConnection) conn).addDataType("geometry",org.postgis.PGgeometry.class);

            loadSchemas();
            jComboBoxMethodItemStateChanged(null);
            config.conn = conn;
            config.tid = "tid";
            config.time = "time";
              }catch(Exception e) {
                        System.out.println(e.toString());
            JOptionPane.showMessageDialog(this,"Error in conection with DB.");
            dispose();
         }
      }

      /**Load the tables in the DB with geometry columns
       */
      private void loadSchemas() {
         try {
            Statement smnt = conn.createStatement();
            ResultSet rs = smnt.executeQuery("SELECT DISTINCT f_table_schema FROM
geometry_columns");
            while (rs.next()) {
               jComboBoxSchema.addItem(rs.getString(1));
            }
         }catch(SQLException e) {
            e.printStackTrace();
            JOptionPane.showMessageDialog(this,"Failed loading schemas");
         }
      }

      private void init() {
            algs = new Method[2];
         int i=0;

         /**SMOT:
                        * Original method used to find stops, given a list of RFs.
                        */
         algs[i] = new Method() {
              public void run2(Trajectory t, InterceptsG in, String targetFeature) throws SQLException {
                        //TrajectoryMethods.smot(jCheckBoxBuffer.isSelected(), buffer, config, t, targetFeature,
relevantFeatures, featureType);
              }
              public void run(Trajectory t,InterceptsG in,String targetFeature,InterceptsG streets) throws
SQLException, IOException {
               java.util.Date ini = new java.util.Date();
               System.out.println("Processing trajectory "+t.tid);
               TrajectoryMethods.smot2(jCheckBoxBuffer.isSelected(),buffer,config,t,
                              jRadioButtonFType.isSelected(),in);

               java.util.Date fim = new java.util.Date();
               java.util.Date tempo = new java.util.Date(fim.getTime()-ini.getTime());
               System.out.println("\tProcessing time: "+tempo.getTime()+" ms");
            }
            public String toString() {
                 return "SMoT";
```

```
        }
    };

    /**CB-SMOT:
     * Used to clusterize and find slow-speed periods in a trajectory.
     */
    algs[++i] = new Method() {
        public void run2(Trajectory t, InterceptsG in, String targetFeature)
                        throws SQLException{}
      public void run(Trajectory t,InterceptsG in,String targetFeature,InterceptsG streets) throws
SQLException {
          //load the Parameter Vector of the method class
         Parameter avg = (Parameter) param.elementAt(0);
          Parameter minTime = (Parameter) param.elementAt(1);
          Parameter speedLimit = (Parameter) param.elementAt(2);

          double SL = ((Double) speedLimit.value).doubleValue();
          int minTimeMili = ((Integer) minTime.value).intValue() * 1000;

          java.util.Date ini = new java.util.Date();

          System.out.println("\t\tStarting Trajectory "+t.tid+"\n\t\tavg= "+((Double)
avg.value).doubleValue()+" ;\n\t\tminTime= "+minTimeMili+" ;\n\t\tSL= "+SL+" ; ");

          // the clustering method, wich will use the points in 't'
          Vector<ClusterPoints> clusters = TrajectoryMethods.speedClustering(t,
                ((Double) avg.value).doubleValue(),
                minTimeMili,
                SL);

          java.util.Date fim = new java.util.Date();
          java.util.Date tempo = new java.util.Date(fim.getTime()-ini.getTime());
          System.out.println("Clusterization: " +tempo.getTime()+" ms");

          //starts to aply semantics...
          ini = new java.util.Date();
          if(jListRF.getMaxSelectionIndex()==-1){//not RF selected
                // save in the stops-table the clusters founded (as unknowns);
                NewTrajectoryMethods.saveStopsClusters(jCheckBoxBuffer.isSelected(),
                                clusters,config,minTimeMili,
                                buffer,t.getSRID(),false);
          }
          else{
                //or attribute semantic given a list of RFs
                 TrajectoryMethods.stopsDiscoveryFaster(jCheckBoxBuffer.isSelected(),
                                buffer,clusters,config,minTimeMili,
                                jRadioButtonFType.isSelected(),in,table_srid,false);
          }
          fim = new java.util.Date();
          tempo = new java.util.Date(fim.getTime()-ini.getTime());
          System.out.println("Semantics Aplication: " +tempo.getTime()+" ms");
        }
        public String toString() {
          return "CB-SMoT";
        }
  };

    algs[i].param.add(
                new Parameter("MaxAvgSpeed",Parameter.Type.DOUBLE,new Double(0.9))
                );
```

```
        algs[i].param.add(
                    new Parameter("MinTime (seconds)",Parameter.Type.INT,new Integer(60))
                    );
        algs[i].param.add(
                    new Parameter("MaxSpeed",Parameter.Type.DOUBLE,new Double(1.1))
                    );
    }

    private void loadTrajectories(String tableTraj) throws SQLException, IOException {
        Method method = (Method) jComboBoxMethod.getSelectedItem();
        InterceptsG i = null;
        InterceptsG streets = null;

        Statement s = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
        // selects the trajectory-tid to be processed
        String sql = "SELECT "+config.tid+" as tid, count(*) FROM "+config.table+" GROUP BY "+config.tid+" ORDER BY tid DESC";
        ResultSet rs = s.executeQuery(sql);

            System.out.print("Creating interceptions...");
            i = createIntercepts();
            System.out.println("Interceptions created ");


        File speedFile = null;
        if(TrajectoryClean.isPrintSpeedToFileXls()){
            speedFile = Util.getFileSpeed(TrajectoryFrame.getCurrentNameTableStop());
        }

            //for each trajectory...
        while (rs.next()) {
            Trajectory trajectory = new Trajectory(table_srid);
            trajectory.tid = rs.getInt("tid");
            String meth = method.toString();

            if (!meth.startsWith("SMoT")) {
              //select the points of the trajectory in sequential time
              Statement s1 = conn.createStatement();
              String sql2 = "SELECT "+config.time+" as time,the_geom,gid FROM "+tableTraj+" WHERE "+config.tid+"="+trajectory.tid+" ORDER BY "+config.time;
              //System.out.println(sql2);

                ResultSet rs2 = s1.executeQuery(sql2);
                //for each of these points of the trajectory...
                int timeIndex = 0;
                while (rs2.next()) {
                    Timestamp t = rs2.getTimestamp("time");
                    org.postgis.PGgeometry geom = (org.postgis.PGgeometry) rs2.getObject("the_geom");
                    org.postgis.Point p = (org.postgis.Point) geom.getGeometry();
                    //get the time, the_geom and tid columns to fill the Vector
                    GPSPoint gps = new GPSPoint(trajectory.tid,t,p,timeIndex);
                    gps.gid = rs2.getInt("gid");
                    trajectory.points.addElement(gps);
                    timeIndex++;
                    /*System.out.println("gid: "+gps.gid+
                                       " tid: "+gps.tid+
                                       " time: "+gps.time+
                                           " timeIndex: "+gps.getTimeIndex());*/
                }
```

```
            //calculates the speed of each point, and then runs the method
                    if (trajectory.points.size()>5){
                            //trajectory.calculatePointsSpeed(2);
                            trajectory.calculatePointsSpeed();
                    double mediaVelocidade  = trajectory.meanSpeed();
                    double mediaDistancia   = trajectory.meanDist();
                    long   duracao          = trajectory.duration();
//                    if(trajectory.tid ==95145){
//                       int uuuui = trajectory.tid;
//                    }else{
//                       continue;
//                    }

                    if(TrajectoryClean.isPrintSpeedToFileXls()){
                       //FIXME: @Hercules. impressao sa velocidade.
                       Util.imprimeVelocidades(trajectory.points, speedFile, rs.isFirst());
                    }
                            method.run(trajectory,i,tableTraj,streets);
                       }
                       else{
                            System.out.println("Trajetory "+trajectory.tid+" has less than 5 points. It will be
desconsidered.");
                       }
          }
          //just runs the method chosen, aplyed to one trajectory
          //also see init();
          else method.run(trajectory,i,tableTraj,streets);


      }
      TrajectoryMethods.resetunknown();
   }

   public static List<Trajectory> getTrajectoriesWithSpeeds(String tableTraj, Config config, Integer
table_srid) throws SQLException, IOException {

      List<Trajectory> trajectorys = new ArrayList<Trajectory>();
      Statement                                                s                                                =
config.conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABL
E);
      // selects the trajectory-tid to be processed
      String sql = "SELECT "+config.tid+" as tid, count(*) FROM "+tableTraj+" GROUP BY
"+config.tid+" ORDER BY tid DESC";
      ResultSet rs = s.executeQuery(sql);

         //for each trajectory...
      while (rs.next()) {
         Trajectory trajectory = null;
         if(table_srid != null){
            trajectory = new Trajectory(table_srid);
         }else{
            trajectory = new Trajectory();
         }
         trajectory.tid = rs.getInt("tid");

         //select the points of the trajectory in sequential time
         Statement s1 = config.conn.createStatement();
         String sql2 = "SELECT "+config.time+" as time,the_geom,gid FROM "+tableTraj+" WHERE
"+config.tid+"="+trajectory.tid+" ORDER BY "+config.time;
```

```
            //System.out.println(sql2);

            ResultSet rs2 = s1.executeQuery(sql2);
            //for each of these points of the trajectory...
            int timeIndex = 0;
            while (rs2.next()) {
               Timestamp t = rs2.getTimestamp("time");
               org.postgis.PGgeometry geom = (org.postgis.PGgeometry) rs2.getObject("the_geom");
               org.postgis.Point p = (org.postgis.Point) geom.getGeometry();
               //get the time, the_geom and tid columns to fill the Vector
               GPSPoint gps = new GPSPoint(trajectory.tid,t,p,timeIndex);
               gps.gid = rs2.getInt("gid");
               trajectory.points.addElement(gps);
               timeIndex++;

            }
            System.out.println("calculando velocidade da trajetoria "+trajectory.tid);
            //calculates the speed of each point, and then runs the method
            if (trajectory.points.size()>5){
               trajectory.calculatePointsSpeed();
            }
            trajectorys.add(trajectory);
            System.out.println("add trajetoria " + trajectory.tid);
            System.gc();
         }
         return trajectorys;
      }



      private void createTables() throws SQLException {
         Statement s = conn.createStatement();
         Statement s1= conn.createStatement();
         Double bufferValue = Double.valueOf(jTextFieldBuffer.getText());

         //@Hercules
         String nmTableStop = nameTableStop(config.table);
         TrajectoryFrame.setCurrentNameTableStop(nmTableStop);
         //@Hercules

         // STOPS table
         System.out.println("\t\tstops table...");//testes...
         try {
            s.execute("DROP TABLE "+TrajectoryFrame.getCurrentNameTableStop());
            s.execute("DELETE FROM geometry_columns WHERE f_table_name =
'"+TrajectoryFrame.getCurrentNameTableStop()+"'");
            //System.out.println("\t\tstops drop...");
         }catch (SQLException ex) {
         }finally {
            s.execute(
               "CREATE TABLE "+TrajectoryFrame.getCurrentNameTableStop()+" ("+
               "    gid serial NOT NULL,"+
               "    tid integer NOT NULL,"+
               "    stopid integer NOT NULL,"+
               "    stop_gid character varying,"+
               "    stop_name character varying,"+
               "    start_time timestamp without time zone,"+
               "    end_time timestamp without time zone,"+
               "    rf character varying,"+
               "    avg real," +
```

```
            "    CONSTRAINT "+TrajectoryFrame.getCurrentNameTableStop()+"_gidkey PRIMARY KEY
(gid)"+
            ") WITHOUT OIDS;"
          );
           s.execute("SELECT    AddGeometryColumn('"+TrajectoryFrame.getCurrentNameTableStop()+"',
'the_geom','"+table_srid+", 'POLYGON', 2)");
          try {
            s.execute("ALTER      TABLE      "+TrajectoryFrame.getCurrentNameTableStop()+"      DROP
CONSTRAINT enforce_geotype_the_geom");
          } catch (SQLException ex) {
            try {
              s.execute("ALTER      TABLE      "+TrajectoryFrame.getCurrentNameTableStop()+"      DROP
CONSTRAINT \"$2\"");
            }catch (SQLException e) {
              ex.printStackTrace();
            }
          }
      }

      //MOVES table (NOT USED YET)
      System.out.println("\t\tmoves table...");
      try {
        s.execute("DROP TABLE moves");
        s.execute("DELETE FROM geometry_columns WHERE f_table_name = 'moves'");
      } catch (SQLException ex) {
      }finally {
        s.execute(
          "CREATE TABLE moves ("+
          "      tid integer NOT NULL,"+
          "      moveid integer NOT NULL,"+
          "      start_time timestamp without time zone,"+
          "      end_time timestamp without time zone,"+
          "      start_stop character varying,"+
          "      end_stop character varying,"+
          "      start_stop_pk integer,"+
          "      end_stop_pk integer,"+
          "      rf character varying,"+
          "      CONSTRAINT moves_pkey PRIMARY KEY (tid,rf,moveid)"+
          ")WITHOUT OIDS"
        );
        s.execute("SELECT   AddGeometryColumn('moves',   'the_geom','"+table_srid+",   'LINESTRING',
2)");
        try {
          s.execute("ALTER TABLE moves DROP CONSTRAINT enforce_geotype_the_geom");
        } catch (SQLException ex) {
          try {
            s.execute("ALTER TABLE moves DROP CONSTRAINT \"$2\"");
          }catch (SQLException e) {
            ex.printStackTrace();
          }
        }
      }

      /*Apply buffer geometry to RFs.  */

      /*
      System.out.println("\t\tenvelopes...");
      Object[] objs = jListRF.getSelectedValues();
      String bufenv,buffer;
      for (Object obj : objs) {
```

```
        String rf = ((AssociatedParameter) obj).name;
        String type = ((AssociatedParameter) obj).name;
        if(!type.contains("POINT")){
                if (jCheckBoxBuffer.isSelected()) {
                    buffer = "buffer(the_geom," + bufferValue + ") as buf,";
                    bufenv = "buffer(envelope(the_geom),"+ bufferValue +") as bufenv";
                }
                // if there's no buffer
                else {
                   buffer = "the_geom as buf,";
                   bufenv = "envelope(the_geom) as bufenv";
                }
                //System.out.println("Here:\n"+buffer+"\n"+bufenv);
                try {
                  String a = "SELECT * FROM "+rf+"_envelope LIMIT 1";
                    s.execute(a);
                }catch (SQLException ex) {
                   //System.out.println(ex.getMessage());
                  try {
                            //do not create a copy in the DB
                            //s.execute("CREATE TEMPORARY TABLE "+rf+"_envelope AS SELECT
gid,"+buffer+bufenv+" FROM "+rf);
                            //create a copy in DB
                            String a = "CREATE TABLE "+rf+"_envelope AS SELECT
gid,"+buffer+bufenv+" FROM "+rf;
                            s.execute(a);
                     } catch (SQLException ex2) {
                            //System.out.println(ex2.getMessage());
                        s.execute("UPDATE "+rf+"_envelope SET env = "+buffer);
                     }
                }
       }// if not POINT
     }
     */

  }

private InterceptsG createIntercepts() throws SQLException{
                    //get the RFs from panel...
          Object[] objs = jListRF.getSelectedValues();
          AssociatedParameter[] relevantFeatures = new AssociatedParameter[objs.length];
     for (int i=0;i<objs.length;i++) {
        relevantFeatures[i] = (AssociatedParameter) objs[i];
     }
     //for each rf, execute the query and save, in main memory, the results
     Statement s = config.conn.createStatement();
     InterceptsG intercs = new InterceptsG();
     for(AssociatedParameter a:relevantFeatures){
          // Create a table of registers with:
          // pt -> gid of the trajectory point
          // gid -> gid from RF wich intercept it
          // rf -> rf_name
          java.util.Date tempo2,fim2,ini2 = new java.util.Date();
          System.out.println("\t\t...with "+a.name);

          String sql;
          if(a.type.contains("POINT") || a.type.contains("LINE")){// if any kind of POINT or LINE
                    try {
              s.execute("DROP TABLE "+a.name+"_buf;");
            } catch (SQLException ex) {
```

```
                    // do nothing
                } finally {
                    s.execute("create table "+a.name+"_buf as select gid, buffer(the_geom,"+buffer+") as
the_geom from "+a.name+";");
                    s.execute("alter table "+a.name+"_buf add constraint "+a.name+"_buf_pk primary key
(gid);");
                }
                    sql=("select A.gid as pt, B.gid as gid, '"+a.name+"' as rf "+
                            "from "+config.table+" A,"+a.name+"_buf B " +
                     "where st_intersects(A.the_geom,B.the_geom);");
            }
            else{
                    /* sql=("select A.gid as pt,B.gid as gid,'"+a.name+"' as rf from "+config.table+"
A,"+a.name+"_envelope B" +
                     " where st_intersects(A.the_geom,B.bufenv) " +
                     "AND st_intersects(A.the_geom,B.buf) ;"); */
                    if (jCheckBoxBuffer.isSelected()) { // if user sets buffer
                            try {
                            s.execute("DROP TABLE "+a.name+"_buf");
                    } catch (SQLException ex) {
                            // do nothing
                    } finally {
                            s.execute("CREATE TABLE "+a.name+"_buf AS SELECT gid,
buffer(the_geom, "+buffer+") AS the_geom FROM "+a.name+";");
                            s.execute("ALTER TABLE "+a.name+"_buf ADD CONSTRAINT
"+a.name+"_buf_pk PRIMARY KEY (gid);");
                    }
                    sql = "select A.gid as pt, B.gid as gid, '"+a.name+"' as rf "+
                        "from "+config.table+" A, "+a.name+"_buf B "+
                        "where st_intersects(A.the_geom,B.the_geom);";
                    } else {
                            sql = "select A.gid as pt, B.gid as gid, '"+a.name+"' as rf "+
                                    "from "+config.table+" A, "+a.name+" B "+
                                    "where st_intersects(A.the_geom,B.the_geom);";
                    }

            }
            ResultSet Intercep = s.executeQuery(sql);
            fim2 = new java.util.Date();
            tempo2 = new java.util.Date(fim2.getTime()-ini2.getTime());
            System.out.println("\t\t"+a.name+" time: " +tempo2.getTime()+" ms");
            // then, save the registers from the query in an adequate struct

            while(Intercep.next()){
              Interc i = new Interc (Intercep.getInt("pt"), Intercep.getInt("gid"),
Intercep.getString("rf"),a.value.intValue());
              intercs.addpt(i);

            }
        }

        return intercs;
}

//-----------------------------------------------------------------------------------------------------------
// INTERFACE BEGINS
//-----------------------------------------------------------------------------------------------------------

    private void initComponents(){
            //Mounts the layout
```

```
                Container container = getContentPane();
          //JPanel container = new JPanel();
                container.setLayout(new BorderLayout());

                //PANEL LOAD SCHEMA
                JPanel panelSchema = new JPanel();
                panelSchema.setBorder(BorderFactory.createEtchedBorder());
                JLabel schemaLabel = new JLabel("Schema:");
                panelSchema.add(schemaLabel,BorderLayout.CENTER);
                jComboBoxSchema = new JComboBox();
                jComboBoxSchema.setPreferredSize(new Dimension(150,22));
                panelSchema.add(jComboBoxSchema,BorderLayout.CENTER);

                JButton Load = new javax.swing.JButton("Load");
                Load.addActionListener(new java.awt.event.ActionListener(){
                   public void actionPerformed(java.awt.event.ActionEvent evt) {
          LoadActionPerformed(evt);
        }});
                panelSchema.add(Load);

                JButton configure = new javax.swing.JButton("Configure Trajectory Table");
                configure.addActionListener(new java.awt.event.ActionListener(){
                   public void actionPerformed(java.awt.event.ActionEvent evt) {
          configureActionPerformed(evt);
        }});
                panelSchema.add(configure);


                //bruno
                JButton show = new javax.swing.JButton("Visualization");
                show.addActionListener(new java.awt.event.ActionListener(){

                        public void actionPerformed(java.awt.event.ActionEvent evt) {
                          showDadosGeograficos();
                        }});
                panelSchema.add(show);
                //bruno


//@Hercules
JButton filter = new javax.swing.JButton("Trajectory cleaning");
filter.addActionListener(new java.awt.event.ActionListener() {

   public void actionPerformed(java.awt.event.ActionEvent evt) {
      filterActionPerformed(evt);
   }
});
panelSchema.add(filter);
//HMA
                container.add(panelSchema,BorderLayout.NORTH);

                //PANEL CONTENT
                JPanel panelContent = new JPanel();
                panelContent.setBorder(BorderFactory.createEtchedBorder());

                GridBagLayout gridbag = new GridBagLayout();
                panelContent.setLayout(gridbag);
                GridBagConstraints c = new GridBagConstraints();

                c.fill = GridBagConstraints.BOTH;
```

```
c.insets = new Insets(5,10,5,10);

        //Config of trajectory Table
        c.gridx=0;
        c.gridy=0;
        JLabel jLabel1 = new javax.swing.JLabel("Trajectory Table: ");
        panelContent.add(jLabel1,c);

        c.gridx=2;
        c.gridwidth=2;
        c.weightx = 1.0;
        JScrollPane sc1 = new javax.swing.JScrollPane();
        DefaultListModel modelsc = new DefaultListModel();
        jListTrajectoryTables= new JList(modelsc);
        jListTrajectoryTables.setVisibleRowCount(2);
        jListTrajectoryTables.setFixedCellWidth(2);
        sc1.setViewportView(jListTrajectoryTables);
        panelContent.add(sc1,c);

        c.gridx=0;
        c.gridy=1;
        c.gridwidth=5;
        c.gridheight=1;
        javax.swing.JButton jButtonConfig = new javax.swing.JButton("Trajectory
Table Config...");


        //Granularity Level-Panel Granularity
        JPanel panelGranularity = new JPanel();
        panelGranularity.setBorder(BorderFactory.createTitledBorder("Granularity
Level"));

            jRadioButtonFType = new JRadioButton("Feature Type",false);
            jRadioButtonFInstance = new JRadioButton("Feature Instance",true);
            ButtonGroup group = new ButtonGroup();
            group.add(jRadioButtonFType);
            group.add(jRadioButtonFInstance);

    panelGranularity.add(jRadioButtonFType);
    panelGranularity.add(jRadioButtonFInstance);

    c.gridx=5;
    c.gridy=0;
    c.gridwidth=5;
    c.gridheight=3;
        panelContent.add(panelGranularity,c);

        //Relevant Features
        c.gridx=0;
        c.gridy=1;
        c.gridheight=2;
        JLabel jLabel2 = new javax.swing.JLabel();
        jLabel2.setText("Relevant Features");
        panelContent.add(jLabel2,c);

        c.gridy=3;
        c.gridwidth=3;
        c.weightx = 1.0;
        JScrollPane jScrollPane1 = new javax.swing.JScrollPane();
        DefaultListModel modelRF = new DefaultListModel();
```

```
                    jListRF= new JList(modelRF);
                    jListRF.setVisibleRowCount(4);
                    jListRF.setFixedCellWidth(4);
                    jListRF.addListSelectionListener(new
javax.swing.event.ListSelectionListener() {
                public void valueChanged(javax.swing.event.ListSelectionEvent evt) {
                  jListRFValueChanged(evt);
                }
              });
                    jScrollPane1.setViewportView(jListRF);
                    panelContent.add(jScrollPane1,c);
                    validate();

                    //Buffer
              JPanel bufPanel = new JPanel();
              bufPanel.setBorder(BorderFactory.createEtchedBorder());
              GridBagLayout gbag = new GridBagLayout();
                    bufPanel.setLayout(gbag);
                    GridBagConstraints c2 = new GridBagConstraints();
                    c2.insets = new Insets(5,5,5,5);

                          c2.gridx=0;
                          c2.gridy=0;
                          jCheckBoxBuffer= new JCheckBox();
                          jCheckBoxBuffer.setSelected(true);
                jCheckBoxBuffer.setText("User Buffer (m):");
                bufPanel.add(jCheckBoxBuffer,c2);

                          c2.gridy=2;
                          c2.gridheight=2;
                          jTextFieldBuffer=new JTextField();
                          jTextFieldBuffer.setPreferredSize(new Dimension(60,20));
                          jTextFieldBuffer.setText("50.0");
                bufPanel.add(jTextFieldBuffer,c2);

              c.gridx=3;
              c.gridy=2;
              c.gridheight=2;
              c.gridwidth=2;
              panelContent.add(bufPanel,c);

              //MinTimeBox
              JPanel mtPanel = new JPanel();
              mtPanel.setBorder(BorderFactory.createEtchedBorder());
              gbag = new GridBagLayout();
                    mtPanel.setLayout(gbag);
                    c2 = new GridBagConstraints();

                c2.gridx=0;
                c2.gridy=0;
                JLabel jLabel3 = new javax.swing.JLabel("RF Min Time (sec): ");
                          mtPanel.add(jLabel3,c2);

                          c2.gridx=0;
                          c2.gridy=1;
                          RFMinTime= new JTextField();
                          RFMinTime.setPreferredSize(new Dimension(40,20));
                          RFMinTime.setColumns(6);
                RFMinTime.addFocusListener(new java.awt.event.FocusAdapter() {
```

```
                    public void focusLost(java.awt.event.FocusEvent evt) {//if 'tab' after entering RF
mintime
                        RFMinTimeFocusLost(evt);
                    }
                });
                RFMinTime.addActionListener(// if 'enter' after entering RF mintime
                                        new ActionListener() {
                                                public void actionPerformed(ActionEvent e)
{
                                                        RFMinTimeActionPerformed(e);
                                                }
                                        }
                                );
                                mtPanel.add(RFMinTime,c2);

                    c.gridx=3;
                    c.gridy=4;
                    panelContent.add(mtPanel,c);

                    //Method Panel
                    JPanel panelMethod = new JPanel();
                    panelMethod.setBorder(BorderFactory.createTitledBorder("Method"));
                    gbag = new GridBagLayout();
                    panelMethod.setLayout(gbag);
                    c2 = new GridBagConstraints();
                    c2.insets=new Insets(3,3,3,3);

                                c2.gridx=0;
                                c2.gridy=0;
                                c2.gridwidth=6;
                                jComboBoxMethod=new JComboBox(algs);
                                jComboBoxMethod.setPreferredSize(new Dimension(210,20));
                                jComboBoxMethod.addItemListener(new
java.awt.event.ItemListener() {
                    public void itemStateChanged(java.awt.event.ItemEvent evt) {
                        jComboBoxMethodItemStateChanged(evt);
                    }
                });
                                panelMethod.add(jComboBoxMethod,c2);

                                c2.gridx=0;
                                c2.gridy=2;
                                c2.gridwidth=3;
                                JLabel jLabel4 = new javax.swing.JLabel("Parameter: ");
                                panelMethod.add(jLabel4,c2);

                                c2.gridx=3;
                                JLabel jLabel5 = new javax.swing.JLabel("Value: ");
                                panelMethod.add(jLabel5,c2);

                                c2.gridx=3;
                                c2.gridy=3;
                                jTextFieldParam= new JTextField();
                                jTextFieldParam.setPreferredSize(new Dimension(40,20));
                                jTextFieldParam.addFocusListener(new java.awt.event.FocusAdapter()
{
                    public void focusLost(java.awt.event.FocusEvent evt) {
                        jTextFieldParamFocusLost(evt);
                    }
                });
```

```
                                panelMethod.add(jTextFieldParam,c2);

                                c2.gridx=0;
                                c2.gridy=3;
                                jComboBoxParam=new JComboBox();
                                jComboBoxParam.setPreferredSize(new Dimension(160,20));
                                jComboBoxParam.addItemListener(new  java.awt.event.ItemListener()
{
                        public void itemStateChanged(java.awt.event.ItemEvent evt) {
                           jComboBoxParamItemStateChanged(evt);
                        }
                     });
                                panelMethod.add(jComboBoxParam,c2);

                   c.gridx=5;
                         c.gridy=3;
                         panelContent.add(panelMethod,c);
                container.add(panelContent,BorderLayout.CENTER);

                JPanel panelDown = new JPanel();
                panelDown.setBorder(BorderFactory.createEtchedBorder());

                gridbag = new GridBagLayout();
                panelDown.setLayout(gridbag);
                c = new GridBagConstraints();

                c.fill = GridBagConstraints.BOTH;
                c.insets = new Insets(10,10,10,10);

                         jButtonGenArffFile = new JButton("Generate Arff File...");
                         jButtonGenArffFile.addActionListener(new java.awt.event.ActionListener() {
              public void actionPerformed(java.awt.event.ActionEvent evt) {
                 jButtonGenArffFileActionPerformed(evt);
              }
           });
                         c.gridx=0;
                         c.gridy=0;
                         panelDown.add(jButtonGenArffFile,c);

                         JButton jButtonOK = new javax.swing.JButton("OK");
                         jButtonOK.addActionListener(new java.awt.event.ActionListener() {
              public void actionPerformed(java.awt.event.ActionEvent evt) {
                 try {
                                                  OKActionPerformed(evt);
                                          } catch (IOException e) {
                                                  e.printStackTrace();
                                                  System.out.println(e.getMessage());
                                          } catch (SQLException e) {
                                                  e.printStackTrace();
                                                  System.out.println(e.getMessage());
                                          }
              }
           });

                         c.gridx=3;
                         c.gridy=0;
                         c.gridwidth=2;
                         panelDown.add(jButtonOK,c);

                         JButton jButtonCancel = new javax.swing.JButton("Close");
```

```
                    jButtonCancel.addActionListener(new java.awt.event.ActionListener() {
                public void actionPerformed(java.awt.event.ActionEvent evt) {
                  jButtonCancelActionPerformed(evt);
                }
              });
                    c.gridx=5;
                    c.gridy=0;
                    c.gridwidth=2;
                    panelDown.add(jButtonCancel,c);

                    c.gridx=9;
                    c.gridy=0;
                    panelDown.add(tm,c);

                container.add(panelDown,BorderLayout.SOUTH);

                setDefaultCloseOperation(DISPOSE_ON_CLOSE);
                this.pack();
                jComboBoxSchema.requestFocusInWindow();
                this.setMaximumSize(new Dimension(600,360));
                this.setSize(680,360);
                this.setVisible(true);
}

private void configureActionPerformed(ActionEvent e) {
        int[] i = jListTrajectoryTables.getSelectedIndices();
        if( i.length == 1){
                Object[] temp = jListTrajectoryTables.getSelectedValues();
                config.table = (String)temp[0];
      config.conn = conn;
      TrajectoryConfig tc = new TrajectoryConfig();
      tc.setConfig(config);
      tc.setVisible(true);
        }
        else{
                JOptionPane.showMessageDialog(this,"Select only one Trajectory Table.");
      return;
        }
}

private void jComboBoxMethodItemStateChanged(java.awt.event.ItemEvent evt) {
   Method alg = (Method) jComboBoxMethod.getSelectedItem();
   jComboBoxParam.removeAllItems();
   for (int i=0;i<alg.param.size();i++) {
      jComboBoxParam.addItem(alg.param.elementAt(i));
   }

   //prevents the SMoT methods to call upon parameters
   if(alg.toString().compareTo("SMoT")==0){
        jComboBoxParam.setEnabled(false);
        jTextFieldParam.setEnabled(false);
   }
   else{
        jComboBoxParam.setEnabled(true);
        jTextFieldParam.setEnabled(true);
   }
}

private boolean checkBufferState() {
   try{
```

```
            buffer = Double.valueOf(jTextFieldBuffer.getText());
            return true;

        } catch (NumberFormatException e){
            jTextFieldBuffer.setText("50.0");
            return false;
        }
    }

    private void jButtonGenArffFileActionPerformed(java.awt.event.ActionEvent evt) {
        GenArffFile gaf = new GenArffFile(conn,jRadioButtonFType.isSelected());
        gaf.setVisible(true);
    }

    private void jButtonCancelActionPerformed(java.awt.event.ActionEvent evt) {
        this.dispose();
    }

    @SuppressWarnings("static-access")
            private void jTextFieldParamFocusLost(java.awt.event.FocusEvent evt) {
        Parameter p = (Parameter) jComboBoxParam.getSelectedItem();
        try {
            if (p.type.DOUBLE == p.type) {
                p.value = new Double(jTextFieldParam.getText());
            }else {
                p.value = new Integer(jTextFieldParam.getText());
            }
        }catch(NumberFormatException e) {
            JOptionPane.showMessageDialog(this,"Parameter value invalid!");
        }
    }

    @SuppressWarnings("static-access")
            private void jComboBoxParamItemStateChanged(java.awt.event.ItemEvent evt) {
        Parameter p = (Parameter) evt.getItem();
        if (p.type == p.type.DOUBLE) {
            jTextFieldParam.setText(((Double)p.value).toString());
        }else {
            jTextFieldParam.setText(((Integer)p.value).toString());
        }
    }

    private void jComboBoxMethodItemStateChanged1(java.awt.event.ItemEvent evt) {
        Method alg = (Method) jComboBoxMethod.getSelectedItem();
        jComboBoxParam.removeAllItems();
        for (int i=0;i<alg.param.size();i++) {
            jComboBoxParam.addItem(alg.param.elementAt(i));
        }
    }

    private void jListRFValueChanged(javax.swing.event.ListSelectionEvent evt) {
        AssociatedParameter par = (AssociatedParameter) jListRF.getSelectedValue();
        if (par != null)
            RFMinTime.setText(""+par.value.intValue());
    }

    private void RFMinTimeFocusLost(java.awt.event.FocusEvent evt) {
        Object[] objs = jListRF.getSelectedValues();
        try {
            for (Object obj : objs) {
```

```
            AssociatedParameter p = (AssociatedParameter) obj;
            p.value = new Integer(Integer.parseInt(RFMinTime.getText()));
          }
          jTextFieldBuffer.grabFocus();
        }catch(java.lang.NumberFormatException e) {
          javax.swing.JOptionPane.showMessageDialog(this, e.toString());
          RFMinTime.grabFocus();
        }
    }

    private void RFMinTimeActionPerformed(java.awt.event.ActionEvent evt) {
        Object[] objs = jListRF.getSelectedValues();
        try {
          for (Object obj : objs) {
            AssociatedParameter p = (AssociatedParameter) obj;
            p.value = new Integer(Integer.parseInt(RFMinTime.getText()));
          }
          jTextFieldBuffer.grabFocus();
        }catch(java.lang.NumberFormatException e) {
          javax.swing.JOptionPane.showMessageDialog(this, e.toString());
          RFMinTime.grabFocus();
        }
    }


    private void LoadActionPerformed(ActionEvent evt) {
           try{ //load the tables in a list of auxiliary strings
          Statement s = conn.createStatement();
          ResultSet vTableName = s.executeQuery("SELECT f_table_name as tableName,type "+
                                    "FROM geometry_columns " +
                                    "WHERE                                    f_table_schema=trim('"+(String)
jComboBoxSchema.getSelectedItem()+"') "+
                                    "ORDER BY tableName");
          DefaultListModel model = (DefaultListModel) jListTrajectoryTables.getModel();//Trajct Tables
list
          model.removeAllElements();
          DefaultListModel model2 = (DefaultListModel) jListRF.getModel();//RF list
          model2.removeAllElements();
          while ( vTableName.next() )  {/* creates a new table for each table that has objects with
topological relation to vRegion */
            model2.addElement(new
AssociatedParameter(vTableName.getString("tableName"),vTableName.getString("type")));// RFs
           model.addElement(new String(vTableName.getString(1)));//Traject tables
          }
        }catch (Exception vErro){
          vErro.printStackTrace();
        }

    }

    private void OKActionPerformed(ActionEvent evt) throws IOException, SQLException {
           if(jCheckBoxBuffer.isSelected()){
                  if(checkBufferState()){
                         System.out.println("Buffer of "+buffer+" saved.");
                  }
                  else{
                         JOptionPane.showMessageDialog(this,"Buffer expects a number.");
                         return;
                  }
           } else {
```

```
                this.buffer = 50.0; // default value
            }

            if (jListRF.getSelectedIndex() == -1 &&
                    jComboBoxMethod.getSelectedItem().toString().compareTo("CB-SMoT")!=0){//cause
CB-SMoT has a version without RFs
             //cause DB-SMoT has a version without RFs too !
           JOptionPane.showMessageDialog(this,"Select one or more relevant features.");
           return;
        }

        if(jListTrajectoryTables.getSelectedIndex() == -1){
            JOptionPane.showMessageDialog(this,"Select one or more trajectory table.");
           return;
        }

        //get the thing in 'things', those trajectories tables to be executed
        Object[] objs = jListTrajectoryTables.getSelectedValues();
        String[] str = new String [objs.length];
        for(int i=0;i<objs.length;i++){
             str[i]=(String)objs[i];
        }

        //controls if SRID of RFs are different from trajectories...
                    // ALL the trajectories should have the SAME srid
                    // it is checked ahead in the foreach.
        config.table = str[0];
        String error = checkSRIDs();//att the variable 'table_srid'
        if(error.compareTo("")!=0){
            JOptionPane.showMessageDialog(this,error);
           return;
        }

        //for each of the trajectory table selected...
        for(int count =0; count<str.length;){
           java.util.Date tempo,fim,ini = new java.util.Date();
           config.table = str[count];
           try {
              //trajectory srid checking, has to be the same of all the other trajectory-talbes and RFs
              Statement sn = conn.createStatement();
              ResultSet rsn = sn.executeQuery("select srid from geometry_columns where
f_table_name='"+config.table+"'");
              rsn.next();
              if(table_srid!=rsn.getInt("srid")){
                 throw new Exception("SRID imcompatibles. Trajectory table "+config.table+" should be
changed.");
              }
              //enf of srid checking
              System.out.println("Creating tables...");
              createTables();
              System.out.println("Processing the trajectories...");

              loadTrajectories(config.table);
              fim = new java.util.Date();
              tempo = new java.util.Date(fim.getTime()-ini.getTime());
              //insertCleanTrajProcess(tempo.getTime());
              count++;
              if (count == str.length){
                 System.out.println("Processing time: " +tempo.getTime()+" ms");
                 JOptionPane.showMessageDialog(this,"Operation finished succesfully.");
```

```
            }
          }catch(Exception e) {
            e.printStackTrace();
            JOptionPane.showMessageDialog(this,"Error during operation");
            System.out.println("Error: \n"+e.getMessage());
          }
          finally{
          }
      }//endof foreach
      Runtime.getRuntime().gc();
   }

   private String checkSRIDs() {

      Statement sn;
                  try {
                        //getting trajectory SRID
                        sn = conn.createStatement();
                        ResultSet rsn = sn.executeQuery("select srid from geometry_columns where
f_table_name='"+config.table+"'");
                rsn.next();
                table_srid=rsn.getInt("srid");

                //getting all the RFs
                Object[] objs = jListRF.getSelectedValues();
                AssociatedParameter[] relevantFeatures = new AssociatedParameter[objs.length];
                for (int i=0;i<objs.length;i++) {
                   relevantFeatures[i] = (AssociatedParameter) objs[i];
                }
                //comparing their SRIDs with the trajectory
                for(AssociatedParameter a:relevantFeatures){
                     sn = conn.createStatement();
                                    rsn = sn.executeQuery("select srid from geometry_columns where
f_table_name='"+config.table+"'");
                           rsn.next();
                           if(table_srid!=rsn.getInt("srid")){
                                return "Error in the SRID of table: "+a.name;
                           }
                }
                     } catch (SQLException e) {
                           System.out.println(e.getMessage());
                           e.printStackTrace();
                     }
          return "";
          }

      //@autor: Bruno
          protected String showDadosGeograficos() {
                  try {
                        //getting trajectory SRID

                        Statement s = conn.createStatement();

               System.out.println("++++++++++++++++++++");
                        if (jListTrajectoryTables.getSelectedIndex() != -1){
                                 ResultSet vTableName = s.executeQuery("SELECT f_table_name as
tableName,type "+
                                                      "FROM geometry_columns " +
                                                      "WHERE          f_table_schema=trim('"+(String)
jComboBoxSchema.getSelectedItem()+"') "+
```

```
                                                "ORDER BY tableName");
                                        int indexAtualVT=0;
                                        while ( vTableName.next() ) {

if(indexAtualVT==jListTrajectoryTables.getSelectedIndex()){

//System.out.println(vTableName.getString("tableName")+"-"+vTableName.getString("type"));

                                                ShowGeoData rep = new ShowGeoData(conn);

                                                JFrame f = new JFrame("Geographic Data
Visualizer");
                                                f.setFocusable(true);
                                                f.requestFocus();
                                                f.getContentPane().setLayout(new BorderLayout());
                                                f.getContentPane().add(rep,
BorderLayout.CENTER);
                                                f.pack();
                                                f.setVisible(true);
                                                f.setSize(new Dimension(800, 600));
                                                f.setResizable(false);

rep.loadPoints(vTableName.getString("tableName"),Color.BLUE,2);
                                                rep.paintAll(f.getGraphics());

//                                              // JUST FOR TESTING
//                                              f.addWindowListener(new  WindowAdapter(){public
void windowClosing(WindowEvent e) {
//                                                      System.exit(0);
//                                                      }});
//                                              // JUST FOR TESTING
                                        }
                                        indexAtualVT++;

                                }

                        }else{
                                System.out.println("Nada selecionado");
                        }

                System.out.println("----------------------");

                   } catch (SQLException e) {
                        System.out.println(e.getMessage());
                        e.printStackTrace();

                   }
                   return "";
           }
      //@autor: Bruno

//-----------------------------------------------------------------------------------------------------------
// INTERFACE ENDS
//-----------------------------------------------------------------------------------------------------------

/*
   public static void main (String args[]){
                   String url = "jdbc:postgresql://localhost:5432/";
                   String db = "";
                   String user = "";
```

```
                    String pass = "" ;
            new TrajectoryFrame(user,pass,url+db);
    }
*/



private void filterActionPerformed(ActionEvent e) {
    String esquema = (String)jComboBoxSchema.getSelectedItem();
    int[] i = jListTrajectoryTables.getSelectedIndices();
    if (i.length >= 1) {
        List<String> listaTab = new ArrayList<String>();
        Object[] temp = jListTrajectoryTables.getSelectedValues();
        for(Object ob: temp){
            listaTab.add((String) ob);
        }
        TrajectoryClean tc = new TrajectoryClean(conn, listaTab);
        tc.setConfig(this.config);
        tc.setEsquema(esquema);
        tc.setVisibleFrame(true);
    } else {
        JOptionPane.showMessageDialog(this, "Select one or more Trajectory Table.");
        return;
    }
}
public String formatNameParameter(String nm){
        System.out.println("Parm = "+nm);
        if(nm.trim().equalsIgnoreCase("MaxAvgSpeed")){
            return "as";
        }else if(nm.trim().equalsIgnoreCase("MinAvgSpeed")){
            return "as";
        }else if(nm.trim().equalsIgnoreCase("MinTime (seconds)")){
            return "mt";
        }else if(nm.trim().equalsIgnoreCase("MaxSpeed")){
            return "ms";
        }else if(nm.trim().equalsIgnoreCase("MinDirChange (degrees)")){
            return "md";
        }else if(nm.trim().equalsIgnoreCase("MaxTolerance (points)")){
            return "mt";
        }else if(nm.trim().equalsIgnoreCase("MinTimeVar (seconds)")){
            return "mtv";
        }else if(nm.trim().equalsIgnoreCase("MinTimeSpeed (seconds)")){
            return "mts";
        }
        return nm;
    }
    public String parametersClusterStr(){
        StringBuilder str = new StringBuilder();
        for(Parameter param :parametersCluster()){
            if(param.name!=null && !param.name.equals("") && param.value!=null){

str.append("_").append(formatNameParameter(param.name)).append("_").append(param.value.toString())
;
            }
        }
        String str1 = str.toString().toLowerCase().replace(".", "_")
                .replace(",", "_").replace(" ", "")
                .replace("(degrees)", "")
                .replace("(seconds)", "")
                .replace("(points)", "");
```

```
    return str1;
}

public List<Parameter> parametersCluster(){
    List<Parameter> list = new ArrayList<Parameter>();
    int tamParans = 0;
    while(tamParans < this.jComboBoxParam.getModel().getSize()){
        Parameter param = (Parameter)this.jComboBoxParam.getModel().getElementAt(tamParans);
        list.add(param);
        tamParans++;
    }
    return list;
}

public String nameTableStop(String sp){
    return "stops_".concat(sp.concat(parametersClusterStr()));
}

private static String currentNameTableStop;

public static String getCurrentNameTableStop() {
    if(currentNameTableStop==null || currentNameTableStop.equals("")){
        return "stops";
    }
    return currentNameTableStop;
}

private static void setCurrentNameTableStop(String currentNameTableStop) {
    TrajectoryFrame.currentNameTableStop = currentNameTableStop;
}

public void insertCleanTrajProcess(long timeProcess){
    Object[] objs = jListRF.getSelectedValues();
    AssociatedParameter[] relevantFeatures = new AssociatedParameter[objs.length];
    Integer rfMinTime = null;
    if(relevantFeatures.length > 0){
        rfMinTime = relevantFeatures[0].value.intValue();
    }
    Double avgSpeed = null;
    Integer minTime = null;
    Double maxSpeed = null;
    int qtidadeStop = 0;

    for(Parameter param: parametersCluster()){
        try{
            if(param.name.equalsIgnoreCase("MaxAvgSpeed")){
                avgSpeed = (Double)param.value;
            }else if(param.name.equalsIgnoreCase("MinAvgSpeed")){
                avgSpeed = (Double)param.value;
            }else if(param.name.equalsIgnoreCase("MinTime (seconds)")){
                minTime = (Integer)param.value;
            }else if(param.name.equalsIgnoreCase("MaxSpeed")){
                maxSpeed = (Double)param.value;
            }
        }catch(Exception e){
            e.printStackTrace();
        }
    }
    String sqlCount = "select count(*) from "+TrajectoryFrame.getCurrentNameTableStop();
    System.out.println(sqlCount.toString());
```

```
        try{
           PreparedStatement psCount = conn.prepareStatement(sqlCount);
           ResultSet resultSet = psCount.executeQuery();
           if(resultSet.next()){
              qtidadeStop = resultSet.getInt(1);
           }
        }catch(SQLException sqle){
           sqle.printStackTrace();
        }
        StringBuilder sql = new StringBuilder();
           sql.append(" INSERT INTO cleantrajprocess( ");
           sql.append("     oidstop, nomestop, tempostop, paramrfmintime, paramminavgspeed,  ");
           sql.append(" parammintime, parammaxspeed, qtidadestop) ");
           sql.append(" VALUES (null, ?, ?, ");
           sql.append(rfMinTime==null?" null,":" ?, ");
           sql.append(avgSpeed ==null?" null,":" ?, ");
           sql.append(minTime  ==null?" null,":" ?, ");
           sql.append(maxSpeed ==null?" null,":" ?, ");
           sql.append(" ? ) ");
        try {
           System.out.println(sql.toString());
           int i=1;
           PreparedStatement ps = conn.prepareStatement(sql.toString());
           ps.setString(i++, TrajectoryFrame.getCurrentNameTableStop());
           ps.setLong  (i++, timeProcess);
           if(rfMinTime!=null){
              ps.setInt   (i++, rfMinTime);
           }
           if(avgSpeed!=null){
              ps.setDouble(i++, avgSpeed);
           }
           if(minTime!=null){
              ps.setInt   (i++, minTime);
           }
           if(maxSpeed!=null){
              ps.setDouble(i++  , maxSpeed);
           }
           ps.setInt(i, qtidadeStop);
           ps.execute();
        } catch (SQLException ex) {
           Logger.getLogger(TrajectoryFrame.class.getName()).log(Level.SEVERE, null, ex);
        }
     }

  public static void main(String args[]) {
     String url = "jdbc:postgresql://localhost:5432/";
     String db = "postgis";
     String user = "postgres";
     String pass = "postgres";
     new TrajectoryFrame(user, pass, url + db);
  }


  }
```

```
package weka.gui.stpm;

import javax.swing.*;

import weka.gui.WekaTaskMonitor;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;
import java.awt.event.*;
import java.io.File;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.List;
import java.util.Vector;
import java.util.logging.Level;
import java.util.logging.Logger;
import weka.gui.geodata.visualizer.ShowGeoData;
import weka.gui.stpm.clean.TrajectoryClean;
import weka.gui.stpm.clean.Util;


//public class TrajectoryFrame extends JFrame{
public class TrajectoryFrame extends JDialog{

        /**set of methods used */
        private Method[] algs;

        /**Connection with the DB*/
        private Connection conn;

        /**User configurations*/
        private Config config = new Config();

        // Method panel
        /**Selects the apropriate method to be aplicated */
    private javax.swing.JComboBox jComboBoxMethod;

    /**Select the parameters of each method */
    private javax.swing.JComboBox jComboBoxParam;

    /**Enters the numbers atributed to each parameter here. */
    private javax.swing.JTextField jTextFieldParam;

    /**The name of the table with streets information. */
    private javax.swing.JComboBox jComboBoxStreet;

    /**The column of that table with information about speed limit (in meters/second). */
    private javax.swing.JComboBox jComboBoxStreetLimit;
```

```
//Relevant Features, Buffer and MinTime
/**List of relevant features, the tables to use to try to discover the places in trajectory.*/
private javax.swing.JList jListRF;

/**Indicates the distance in meters of the buffer in the relevant features. */
//private javax.swing.JSpinner jSpinnerBuffer;
private javax.swing.JTextField jTextFieldBuffer;

/**Use or not buffer in the relevant features */
private javax.swing.JCheckBox jCheckBoxBuffer;

/**Text field for entering the min time. Must enter the seconds and press 'TAB', for recording. */
        private javax.swing.JTextField RFMinTime;

//Feature type/instance
/**If selected, says that we are working with Feature Instance. */
private javax.swing.JRadioButton jRadioButtonFInstance;

/**If selected, says that we are working with Feature Type. */
private javax.swing.JRadioButton jRadioButtonFType;

//others...
/**Opens the Generate Arff File Frame */
private javax.swing.JButton jButtonGenArffFile;

/** trying to make the bird to fly... */
private final WekaTaskMonitor tm = new WekaTaskMonitor();

/**The thing (trajectories tables) which we want to aply the method selected*/
private javax.swing.JList jListTrajectoryTables; // replaces the jComboBoxTF

/**Select the schema of DB, usually 'public' for localhost tests. */
private javax.swing.JComboBox jComboBoxSchema;

/**DEPRECATED*/
private javax.swing.JComboBox jComboBoxTF; //old targetfeature combbox

private javax.swing.ButtonGroup buttonGroup1;//used in the genarfffile frame
private javax.swing.ButtonGroup buttonGroupItem;// the same
private javax.swing.ButtonGroup buttonGroupTime;//the same

/**Spatial reference for ALL trajectory_table*/
private int table_srid; //spatial reference ID,google it

/**Buffer variable*/
private Double buffer=50.0;       // variable buffer, initialized

public TrajectoryFrame(){
        this.setTitle("Trajectory");
   init();
        initComponents();
}

public TrajectoryFrame(String user, String pass, String url) {
        this.setTitle("Trajectory");
        init();
   initComponents();
        try {
      if (user == "")
```

```
            conn = DriverManager.getConnection(url);
        else
            conn = DriverManager.getConnection(url,user,pass);

        ((org.postgresql.PGConnection) conn).addDataType("geometry",org.postgis.PGgeometry.class);

        loadSchemas();
        jComboBoxMethodItemStateChanged(null);
        config.conn = conn;
        config.tid = "tid";
        config.time = "time";
          }catch(Exception e) {
                System.out.println(e.toString());
        JOptionPane.showMessageDialog(this,"Error in conection with DB.");
        dispose();
      }
   }

   /**Load the tables in the DB with geometry columns
    */
   private void loadSchemas() {
      try {
         Statement smnt = conn.createStatement();
         ResultSet rs = smnt.executeQuery("SELECT DISTINCT f_table_schema FROM
geometry_columns");
         while (rs.next()) {
            jComboBoxSchema.addItem(rs.getString(1));
         }
      }catch(SQLException e) {
         e.printStackTrace();
         JOptionPane.showMessageDialog(this,"Failed loading schemas");
      }
   }

   private void init() {
         algs = new Method[2];
      int i=0;

      /**SMOT:
                * Original method used to find stops, given a list of RFs.
                */
      algs[i] = new Method() {
          public void run2(Trajectory t, InterceptsG in, String targetFeature) throws SQLException {
                //TrajectoryMethods.smot(jCheckBoxBuffer.isSelected(), buffer, config, t, targetFeature,
relevantFeatures, featureType);
          }
          public void run(Trajectory t,InterceptsG in,String targetFeature,InterceptsG streets) throws
SQLException, IOException {
            java.util.Date ini = new java.util.Date();
            System.out.println("Processing trajectory "+t.tid);
            TrajectoryMethods.smot2(jCheckBoxBuffer.isSelected(),buffer,config,t,
                        jRadioButtonFType.isSelected(),in);

            java.util.Date fim = new java.util.Date();
            java.util.Date tempo = new java.util.Date(fim.getTime()-ini.getTime());
            System.out.println("\tProcessing time: "+tempo.getTime()+" ms");
        }
        public String toString() {
            return "SMoT";
        }
```

```
    };

    /**CB-SMOT:
     * Used to clusterize and find slow-speed periods in a trajectory.
     */
    algs[++i] = new Method() {
        public void run2(Trajectory t, InterceptsG in, String targetFeature)
                        throws SQLException{}
      public void run(Trajectory t,InterceptsG in,String targetFeature,InterceptsG streets) throws
SQLException {
          //load the Parameter Vector of the method class
         Parameter avg = (Parameter) param.elementAt(0);
          Parameter minTime = (Parameter) param.elementAt(1);
          Parameter speedLimit = (Parameter) param.elementAt(2);

          double SL = ((Double) speedLimit.value).doubleValue();
          int minTimeMili = ((Integer) minTime.value).intValue() * 1000;

          java.util.Date ini = new java.util.Date();

          System.out.println("\t\tStarting Trajectory "+t.tid+"\n\t\tavg= "+((Double)
avg.value).doubleValue()+" ;\n\t\tminTime= "+minTimeMili+" ;\n\t\tSL= "+SL+" ; ");

          // the clustering method, wich will use the points in 't'
          Vector<ClusterPoints> clusters = TrajectoryMethods.speedClustering(t,
              ((Double) avg.value).doubleValue(),
              minTimeMili,
              SL);

          java.util.Date fim = new java.util.Date();
          java.util.Date tempo = new java.util.Date(fim.getTime()-ini.getTime());
          System.out.println("Clusterization: " +tempo.getTime()+" ms");

          //starts to aply semantics...
          ini = new java.util.Date();
          if(jListRF.getMaxSelectionIndex()==-1){//not RF selected
                // save in the stops-table the clusters founded (as unknowns);
                NewTrajectoryMethods.saveStopsClusters(jCheckBoxBuffer.isSelected(),
                                clusters,config,minTimeMili,
                                buffer,t.getSRID(),false);
          }
          else{
                //or attribute semantic given a list of RFs
                 TrajectoryMethods.stopsDiscoveryFaster(jCheckBoxBuffer.isSelected(),
                                buffer,clusters,config,minTimeMili,
                                jRadioButtonFType.isSelected(),in,table_srid,false);
          }
          fim = new java.util.Date();
          tempo = new java.util.Date(fim.getTime()-ini.getTime());
          System.out.println("Semantics Aplication: " +tempo.getTime()+" ms");
        }
        public String toString() {
          return "CB-SMoT";
        }
   };

    algs[i].param.add(
              new Parameter("MaxAvgSpeed",Parameter.Type.DOUBLE,new Double(0.9))
              );
    algs[i].param.add(
```

```
                new Parameter("MinTime (seconds)",Parameter.Type.INT,new Integer(60))
                );
        algs[i].param.add(
                new Parameter("MaxSpeed",Parameter.Type.DOUBLE,new Double(1.1))
                );
    }

    private void loadTrajectories(String tableTraj) throws SQLException, IOException {
        Method method = (Method) jComboBoxMethod.getSelectedItem();
        InterceptsG i = null;
        InterceptsG streets = null;

        Statement s = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
        // selects the trajectory-tid to be processed
        String sql = "SELECT "+config.tid+" as tid, count(*) FROM "+config.table+" GROUP BY "+config.tid+" ORDER BY tid DESC";
        ResultSet rs = s.executeQuery(sql);

            System.out.print("Creating interceptions...");
            i = createIntercepts();
            System.out.println("Interceptions created ");


        File speedFile = null;
        if(TrajectoryClean.isPrintSpeedToFileXls()){
          speedFile = Util.getFileSpeed(TrajectoryFrame.getCurrentNameTableStop());
        }

            //for each trajectory...
        while (rs.next()) {
            Trajectory trajectory = new Trajectory(table_srid);
          trajectory.tid = rs.getInt("tid");
          String meth = method.toString();

          if (!meth.startsWith("SMoT")) {
            //select the points of the trajectory in sequential time
            Statement s1 = conn.createStatement();
            String sql2 = "SELECT "+config.time+" as time,the_geom,gid FROM "+tableTraj+" WHERE "+config.tid+"="+trajectory.tid+" ORDER BY "+config.time;
            //System.out.println(sql2);

              ResultSet rs2 = s1.executeQuery(sql2);
              //for each of these points of the trajectory...
              int timeIndex = 0;
              while (rs2.next()) {
                 Timestamp t = rs2.getTimestamp("time");
                 org.postgis.PGgeometry geom = (org.postgis.PGgeometry) rs2.getObject("the_geom");
                 org.postgis.Point p = (org.postgis.Point) geom.getGeometry();
                 //get the time, the_geom and tid columns to fill the Vector
                 GPSPoint gps = new GPSPoint(trajectory.tid,t,p,timeIndex);
                 gps.gid = rs2.getInt("gid");
                 trajectory.points.addElement(gps);
                 timeIndex++;
                 /*System.out.println("gid: "+gps.gid+
                                    " tid: "+gps.tid+
                                    " time: "+gps.time+
                                        " timeIndex: "+gps.getTimeIndex());*/
              }
```

```
            //calculates the speed of each point, and then runs the method
                    if (trajectory.points.size()>5){
                            //trajectory.calculatePointsSpeed(2);
                            trajectory.calculatePointsSpeed();
                    double mediaVelocidade  = trajectory.meanSpeed();
                    double mediaDistancia   = trajectory.meanDist();
                    long   duracao          = trajectory.duration();
//                    if(trajectory.tid ==95145){
//                       int uuuui = trajectory.tid;
//                    }else{
//                       continue;
//                    }

                    if(TrajectoryClean.isPrintSpeedToFileXls()){
                       //FIXME: @Hercules. impressao sa velocidade.
                       Util.imprimeVelocidades(trajectory.points, speedFile, rs.isFirst());
                    }
                            method.run(trajectory,i,tableTraj,streets);
                       }
                       else{
                            System.out.println("Trajetory "+trajectory.tid+" has less than 5 points. It will be
desconsidered.");
                       }
            }
            //just runs the method chosen, aplyed to one trajectory
            //also see init();
            else method.run(trajectory,i,tableTraj,streets);


        }
        TrajectoryMethods.resetunknown();
    }

    public static List<Trajectory> getTrajectoriesWithSpeeds(String tableTraj, Config config, Integer
table_srid) throws SQLException, IOException {

        List<Trajectory> trajectorys = new ArrayList<Trajectory>();
        Statement                                                s                                                =
config.conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABL
E);
        // selects the trajectory-tid to be processed
        String sql = "SELECT "+config.tid+" as tid, count(*) FROM "+tableTraj+" GROUP BY
"+config.tid+" ORDER BY tid DESC";
        ResultSet rs = s.executeQuery(sql);

            //for each trajectory...
        while (rs.next()) {
            Trajectory trajectory = null;
            if(table_srid != null){
               trajectory = new Trajectory(table_srid);
            }else{
               trajectory = new Trajectory();
            }
            trajectory.tid = rs.getInt("tid");

            //select the points of the trajectory in sequential time
            Statement s1 = config.conn.createStatement();
            String sql2 = "SELECT "+config.time+" as time,the_geom,gid FROM "+tableTraj+" WHERE
"+config.tid+"="+trajectory.tid+" ORDER BY "+config.time;
            //System.out.println(sql2);
```

```
            ResultSet rs2 = s1.executeQuery(sql2);
            //for each of these points of the trajectory...
            int timeIndex = 0;
            while (rs2.next()) {
               Timestamp t = rs2.getTimestamp("time");
               org.postgis.PGgeometry geom = (org.postgis.PGgeometry) rs2.getObject("the_geom");
               org.postgis.Point p = (org.postgis.Point) geom.getGeometry();
               //get the time, the_geom and tid columns to fill the Vector
               GPSPoint gps = new GPSPoint(trajectory.tid,t,p,timeIndex);
               gps.gid = rs2.getInt("gid");
               trajectory.points.addElement(gps);
               timeIndex++;

            }
            System.out.println("calculando velocidade da trajetoria "+trajectory.tid);
            //calculates the speed of each point, and then runs the method
            if (trajectory.points.size()>5){
               trajectory.calculatePointsSpeed();
            }
            trajectorys.add(trajectory);
            System.out.println("add trajetoria " + trajectory.tid);
            System.gc();
         }
         return trajectorys;
      }



      private void createTables() throws SQLException {
         Statement s = conn.createStatement();
         Statement s1= conn.createStatement();
         Double bufferValue = Double.valueOf(jTextFieldBuffer.getText());

         //@Hercules
         String nmTableStop = nameTableStop(config.table);
         TrajectoryFrame.setCurrentNameTableStop(nmTableStop);
         //@Hercules

         // STOPS table
         System.out.println("\t\tstops table...");//testes...
         try {
            s.execute("DROP TABLE "+TrajectoryFrame.getCurrentNameTableStop());
            s.execute("DELETE FROM geometry_columns WHERE f_table_name =
'"+TrajectoryFrame.getCurrentNameTableStop()+"'");
            //System.out.println("\t\tstops drop...");
         }catch (SQLException ex) {
         }finally {
            s.execute(
               "CREATE TABLE "+TrajectoryFrame.getCurrentNameTableStop()+" ("+
               "    gid serial NOT NULL,"+
               "    tid integer NOT NULL,"+
               "    stopid integer NOT NULL,"+
               "    stop_gid character varying,"+
               "    stop_name character varying,"+
               "    start_time timestamp without time zone,"+
               "    end_time timestamp without time zone,"+
               "    rf character varying,"+
               "    avg real," +
```

```
            "    CONSTRAINT "+TrajectoryFrame.getCurrentNameTableStop()+"_gidkey PRIMARY KEY
(gid)"+
            ") WITHOUT OIDS;"
          );
           s.execute("SELECT    AddGeometryColumn('"+TrajectoryFrame.getCurrentNameTableStop()+"',
'the_geom','"+table_srid+", 'POLYGON', 2)");
          try {
            s.execute("ALTER      TABLE      "+TrajectoryFrame.getCurrentNameTableStop()+"      DROP
CONSTRAINT enforce_geotype_the_geom");
          } catch (SQLException ex) {
            try {
              s.execute("ALTER      TABLE      "+TrajectoryFrame.getCurrentNameTableStop()+"      DROP
CONSTRAINT \"$2\"");
            }catch (SQLException e) {
              ex.printStackTrace();
            }
          }
      }

      //MOVES table (NOT USED YET)
      System.out.println("\t\tmoves table...");
      try {
        s.execute("DROP TABLE moves");
        s.execute("DELETE FROM geometry_columns WHERE f_table_name = 'moves'");
      } catch (SQLException ex) {
      }finally {
        s.execute(
          "CREATE TABLE moves ("+
          "      tid integer NOT NULL,"+
          "      moveid integer NOT NULL,"+
          "      start_time timestamp without time zone,"+
          "      end_time timestamp without time zone,"+
          "      start_stop character varying,"+
          "      end_stop character varying,"+
          "      start_stop_pk integer,"+
          "      end_stop_pk integer,"+
          "      rf character varying,"+
          "      CONSTRAINT moves_pkey PRIMARY KEY (tid,rf,moveid)"+
          ")WITHOUT OIDS"
        );
        s.execute("SELECT   AddGeometryColumn('moves',   'the_geom','"+table_srid+",   'LINESTRING',
2)");
        try {
          s.execute("ALTER TABLE moves DROP CONSTRAINT enforce_geotype_the_geom");
        } catch (SQLException ex) {
          try {
            s.execute("ALTER TABLE moves DROP CONSTRAINT \"$2\"");
          }catch (SQLException e) {
            ex.printStackTrace();
          }
        }
      }

      /*Apply buffer geometry to RFs.  */

      /*
      System.out.println("\t\tenvelopes...");
      Object[] objs = jListRF.getSelectedValues();
      String bufenv,buffer;
      for (Object obj : objs) {
```

```
        String rf = ((AssociatedParameter) obj).name;
        String type = ((AssociatedParameter) obj).name;
        if(!type.contains("POINT")){
                if (jCheckBoxBuffer.isSelected()) {
                    buffer = "buffer(the_geom," + bufferValue + ") as buf,";
                    bufenv = "buffer(envelope(the_geom),"+ bufferValue +") as bufenv";
                }
                // if there's no buffer
                else {
                   buffer = "the_geom as buf,";
                   bufenv = "envelope(the_geom) as bufenv";
                }
                //System.out.println("Here:\n"+buffer+"\n"+bufenv);
                try {
                  String a = "SELECT * FROM "+rf+"_envelope LIMIT 1";
                    s.execute(a);
                }catch (SQLException ex) {
                   //System.out.println(ex.getMessage());
                  try {
                            //do not create a copy in the DB
                            //s.execute("CREATE TEMPORARY TABLE "+rf+"_envelope AS SELECT
gid,"+buffer+bufenv+" FROM "+rf);
                            //create a copy in DB
                            String a = "CREATE TABLE "+rf+"_envelope AS SELECT
gid,"+buffer+bufenv+" FROM "+rf;
                            s.execute(a);
                     } catch (SQLException ex2) {
                            //System.out.println(ex2.getMessage());
                        s.execute("UPDATE "+rf+"_envelope SET env = "+buffer);
                     }
                }
       }// if not POINT
     }
     */

  }

private InterceptsG createIntercepts() throws SQLException{
                    //get the RFs from panel...
          Object[] objs = jListRF.getSelectedValues();
          AssociatedParameter[] relevantFeatures = new AssociatedParameter[objs.length];
     for (int i=0;i<objs.length;i++) {
        relevantFeatures[i] = (AssociatedParameter) objs[i];
     }
     //for each rf, execute the query and save, in main memory, the results
     Statement s = config.conn.createStatement();
     InterceptsG intercs = new InterceptsG();
     for(AssociatedParameter a:relevantFeatures){
          // Create a table of registers with:
          // pt -> gid of the trajectory point
          // gid -> gid from RF wich intercept it
          // rf -> rf_name
          java.util.Date tempo2,fim2,ini2 = new java.util.Date();
          System.out.println("\t\t...with "+a.name);

          String sql;
          if(a.type.contains("POINT") || a.type.contains("LINE")){// if any kind of POINT or LINE
                    try {
              s.execute("DROP TABLE "+a.name+"_buf;");
            } catch (SQLException ex) {
```

```
                    // do nothing
                } finally {
                    s.execute("create table "+a.name+"_buf as select gid, buffer(the_geom,"+buffer+") as
the_geom from "+a.name+";");
                    s.execute("alter table "+a.name+"_buf add constraint "+a.name+"_buf_pk primary key
(gid);");
                }
                    sql=("select A.gid as pt, B.gid as gid, '"+a.name+"' as rf "+
                            "from "+config.table+" A,"+a.name+"_buf B " +
                     "where st_intersects(A.the_geom,B.the_geom);");
            }
            else{
                    /* sql=("select A.gid as pt,B.gid as gid,'"+a.name+"' as rf from "+config.table+"
A,"+a.name+"_envelope B" +
                     " where st_intersects(A.the_geom,B.bufenv) " +
                     "AND st_intersects(A.the_geom,B.buf) ;"); */
                    if (jCheckBoxBuffer.isSelected()) { // if user sets buffer
                            try {
                            s.execute("DROP TABLE "+a.name+"_buf");
                    } catch (SQLException ex) {
                            // do nothing
                    } finally {
                            s.execute("CREATE TABLE "+a.name+"_buf AS SELECT gid,
buffer(the_geom, "+buffer+") AS the_geom FROM "+a.name+";");
                            s.execute("ALTER TABLE "+a.name+"_buf ADD CONSTRAINT
"+a.name+"_buf_pk PRIMARY KEY (gid);");
                    }
                    sql = "select A.gid as pt, B.gid as gid, '"+a.name+"' as rf "+
                        "from "+config.table+" A, "+a.name+"_buf B "+
                        "where st_intersects(A.the_geom,B.the_geom);";
                    } else {
                            sql = "select A.gid as pt, B.gid as gid, '"+a.name+"' as rf "+
                                    "from "+config.table+" A, "+a.name+" B "+
                                    "where st_intersects(A.the_geom,B.the_geom);";
                    }

            }
            ResultSet Intercep = s.executeQuery(sql);
            fim2 = new java.util.Date();
            tempo2 = new java.util.Date(fim2.getTime()-ini2.getTime());
            System.out.println("\t\t"+a.name+" time: " +tempo2.getTime()+" ms");
            // then, save the registers from the query in an adequate struct

            while(Intercep.next()){
              Interc i = new Interc (Intercep.getInt("pt"), Intercep.getInt("gid"),
Intercep.getString("rf"),a.value.intValue());
              intercs.addpt(i);

            }
        }

        return intercs;
}

//-----------------------------------------------------------------------------------------------------------
// INTERFACE BEGINS
//-----------------------------------------------------------------------------------------------------------

    private void initComponents(){
            //Mounts the layout
```

```
                Container container = getContentPane();
          //JPanel container = new JPanel();
                container.setLayout(new BorderLayout());

                //PANEL LOAD SCHEMA
                JPanel panelSchema = new JPanel();
                panelSchema.setBorder(BorderFactory.createEtchedBorder());
                JLabel schemaLabel = new JLabel("Schema:");
                panelSchema.add(schemaLabel,BorderLayout.CENTER);
                jComboBoxSchema = new JComboBox();
                jComboBoxSchema.setPreferredSize(new Dimension(150,22));
                panelSchema.add(jComboBoxSchema,BorderLayout.CENTER);

                JButton Load = new javax.swing.JButton("Load");
                Load.addActionListener(new java.awt.event.ActionListener(){
                   public void actionPerformed(java.awt.event.ActionEvent evt) {
          LoadActionPerformed(evt);
        }});
                panelSchema.add(Load);

                JButton configure = new javax.swing.JButton("Configure Trajectory Table");
                configure.addActionListener(new java.awt.event.ActionListener(){
                   public void actionPerformed(java.awt.event.ActionEvent evt) {
          configureActionPerformed(evt);
        }});
                panelSchema.add(configure);


                //bruno
                JButton show = new javax.swing.JButton("Visualization");
                show.addActionListener(new java.awt.event.ActionListener(){

                        public void actionPerformed(java.awt.event.ActionEvent evt) {
                          showDadosGeograficos();
                        }});
                panelSchema.add(show);
                //bruno


//@Hercules
JButton filter = new javax.swing.JButton("Trajectory cleaning");
filter.addActionListener(new java.awt.event.ActionListener() {

   public void actionPerformed(java.awt.event.ActionEvent evt) {
      filterActionPerformed(evt);
   }
});
panelSchema.add(filter);
//HMA
                container.add(panelSchema,BorderLayout.NORTH);

                //PANEL CONTENT
                JPanel panelContent = new JPanel();
                panelContent.setBorder(BorderFactory.createEtchedBorder());

                GridBagLayout gridbag = new GridBagLayout();
                panelContent.setLayout(gridbag);
                GridBagConstraints c = new GridBagConstraints();

                c.fill = GridBagConstraints.BOTH;
```

```
c.insets = new Insets(5,10,5,10);

        //Config of trajectory Table
        c.gridx=0;
        c.gridy=0;
        JLabel jLabel1 = new javax.swing.JLabel("Trajectory Table: ");
        panelContent.add(jLabel1,c);

        c.gridx=2;
        c.gridwidth=2;
        c.weightx = 1.0;
        JScrollPane sc1 = new javax.swing.JScrollPane();
        DefaultListModel modelsc = new DefaultListModel();
        jListTrajectoryTables= new JList(modelsc);
        jListTrajectoryTables.setVisibleRowCount(2);
        jListTrajectoryTables.setFixedCellWidth(2);
        sc1.setViewportView(jListTrajectoryTables);
        panelContent.add(sc1,c);

        c.gridx=0;
        c.gridy=1;
        c.gridwidth=5;
        c.gridheight=1;
        javax.swing.JButton jButtonConfig = new javax.swing.JButton("Trajectory
Table Config...");


        //Granularity Level-Panel Granularity
        JPanel panelGranularity = new JPanel();
        panelGranularity.setBorder(BorderFactory.createTitledBorder("Granularity
Level"));

            jRadioButtonFType = new JRadioButton("Feature Type",false);
            jRadioButtonFInstance = new JRadioButton("Feature Instance",true);
            ButtonGroup group = new ButtonGroup();
            group.add(jRadioButtonFType);
            group.add(jRadioButtonFInstance);

    panelGranularity.add(jRadioButtonFType);
    panelGranularity.add(jRadioButtonFInstance);

    c.gridx=5;
    c.gridy=0;
    c.gridwidth=5;
    c.gridheight=3;
        panelContent.add(panelGranularity,c);

        //Relevant Features
        c.gridx=0;
        c.gridy=1;
        c.gridheight=2;
        JLabel jLabel2 = new javax.swing.JLabel();
        jLabel2.setText("Relevant Features");
        panelContent.add(jLabel2,c);

        c.gridy=3;
        c.gridwidth=3;
        c.weightx = 1.0;
        JScrollPane jScrollPane1 = new javax.swing.JScrollPane();
        DefaultListModel modelRF = new DefaultListModel();
```

```
                    jListRF= new JList(modelRF);
                    jListRF.setVisibleRowCount(4);
                    jListRF.setFixedCellWidth(4);
                    jListRF.addListSelectionListener(new
javax.swing.event.ListSelectionListener() {
                public void valueChanged(javax.swing.event.ListSelectionEvent evt) {
                  jListRFValueChanged(evt);
                }
              });
                    jScrollPane1.setViewportView(jListRF);
                    panelContent.add(jScrollPane1,c);
                    validate();

                    //Buffer
              JPanel bufPanel = new JPanel();
              bufPanel.setBorder(BorderFactory.createEtchedBorder());
              GridBagLayout gbag = new GridBagLayout();
                    bufPanel.setLayout(gbag);
                    GridBagConstraints c2 = new GridBagConstraints();
                    c2.insets = new Insets(5,5,5,5);

                          c2.gridx=0;
                          c2.gridy=0;
                          jCheckBoxBuffer= new JCheckBox();
                          jCheckBoxBuffer.setSelected(true);
                jCheckBoxBuffer.setText("User Buffer (m):");
                bufPanel.add(jCheckBoxBuffer,c2);

                          c2.gridy=2;
                          c2.gridheight=2;
                          jTextFieldBuffer=new JTextField();
                          jTextFieldBuffer.setPreferredSize(new Dimension(60,20));
                          jTextFieldBuffer.setText("50.0");
                bufPanel.add(jTextFieldBuffer,c2);

              c.gridx=3;
              c.gridy=2;
              c.gridheight=2;
              c.gridwidth=2;
              panelContent.add(bufPanel,c);

              //MinTimeBox
              JPanel mtPanel = new JPanel();
              mtPanel.setBorder(BorderFactory.createEtchedBorder());
              gbag = new GridBagLayout();
                    mtPanel.setLayout(gbag);
                    c2 = new GridBagConstraints();

                c2.gridx=0;
                c2.gridy=0;
                JLabel jLabel3 = new javax.swing.JLabel("RF Min Time (sec): ");
                          mtPanel.add(jLabel3,c2);

                          c2.gridx=0;
                          c2.gridy=1;
                          RFMinTime= new JTextField();
                          RFMinTime.setPreferredSize(new Dimension(40,20));
                          RFMinTime.setColumns(6);
                RFMinTime.addFocusListener(new java.awt.event.FocusAdapter() {
```

```
                    public void focusLost(java.awt.event.FocusEvent evt) {//if 'tab' after entering RF
mintime
                        RFMinTimeFocusLost(evt);
                    }
                });
                RFMinTime.addActionListener(// if 'enter' after entering RF mintime
                                        new ActionListener() {
                                                public void actionPerformed(ActionEvent e)
{
                                                        RFMinTimeActionPerformed(e);
                                                }
                                        }
                                );
                                mtPanel.add(RFMinTime,c2);

                    c.gridx=3;
                    c.gridy=4;
                    panelContent.add(mtPanel,c);

                    //Method Panel
                    JPanel panelMethod = new JPanel();
                    panelMethod.setBorder(BorderFactory.createTitledBorder("Method"));
                    gbag = new GridBagLayout();
                    panelMethod.setLayout(gbag);
                    c2 = new GridBagConstraints();
                    c2.insets=new Insets(3,3,3,3);

                                c2.gridx=0;
                                c2.gridy=0;
                                c2.gridwidth=6;
                                jComboBoxMethod=new JComboBox(algs);
                                jComboBoxMethod.setPreferredSize(new Dimension(210,20));
                                jComboBoxMethod.addItemListener(new
java.awt.event.ItemListener() {
                    public void itemStateChanged(java.awt.event.ItemEvent evt) {
                        jComboBoxMethodItemStateChanged(evt);
                    }
                });
                                panelMethod.add(jComboBoxMethod,c2);

                                c2.gridx=0;
                                c2.gridy=2;
                                c2.gridwidth=3;
                                JLabel jLabel4 = new javax.swing.JLabel("Parameter: ");
                                panelMethod.add(jLabel4,c2);

                                c2.gridx=3;
                                JLabel jLabel5 = new javax.swing.JLabel("Value: ");
                                panelMethod.add(jLabel5,c2);

                                c2.gridx=3;
                                c2.gridy=3;
                                jTextFieldParam= new JTextField();
                                jTextFieldParam.setPreferredSize(new Dimension(40,20));
                                jTextFieldParam.addFocusListener(new java.awt.event.FocusAdapter()
{
                    public void focusLost(java.awt.event.FocusEvent evt) {
                        jTextFieldParamFocusLost(evt);
                    }
                });
```

```
                                panelMethod.add(jTextFieldParam,c2);

                                c2.gridx=0;
                                c2.gridy=3;
                                jComboBoxParam=new JComboBox();
                                jComboBoxParam.setPreferredSize(new Dimension(160,20));
                                jComboBoxParam.addItemListener(new  java.awt.event.ItemListener()
{
                        public void itemStateChanged(java.awt.event.ItemEvent evt) {
                           jComboBoxParamItemStateChanged(evt);
                        }
                     });
                                panelMethod.add(jComboBoxParam,c2);

                   c.gridx=5;
                         c.gridy=3;
                         panelContent.add(panelMethod,c);
                container.add(panelContent,BorderLayout.CENTER);

                JPanel panelDown = new JPanel();
                panelDown.setBorder(BorderFactory.createEtchedBorder());

                gridbag = new GridBagLayout();
                panelDown.setLayout(gridbag);
                c = new GridBagConstraints();

                c.fill = GridBagConstraints.BOTH;
                c.insets = new Insets(10,10,10,10);

                         jButtonGenArffFile = new JButton("Generate Arff File...");
                         jButtonGenArffFile.addActionListener(new java.awt.event.ActionListener() {
              public void actionPerformed(java.awt.event.ActionEvent evt) {
                 jButtonGenArffFileActionPerformed(evt);
              }
           });
                         c.gridx=0;
                         c.gridy=0;
                         panelDown.add(jButtonGenArffFile,c);

                         JButton jButtonOK = new javax.swing.JButton("OK");
                         jButtonOK.addActionListener(new java.awt.event.ActionListener() {
              public void actionPerformed(java.awt.event.ActionEvent evt) {
                 try {
                                                  OKActionPerformed(evt);
                                          } catch (IOException e) {
                                                  e.printStackTrace();
                                                  System.out.println(e.getMessage());
                                          } catch (SQLException e) {
                                                  e.printStackTrace();
                                                  System.out.println(e.getMessage());
                                          }
              }
           });

                         c.gridx=3;
                         c.gridy=0;
                         c.gridwidth=2;
                         panelDown.add(jButtonOK,c);

                         JButton jButtonCancel = new javax.swing.JButton("Close");
```

```
                    jButtonCancel.addActionListener(new java.awt.event.ActionListener() {
                public void actionPerformed(java.awt.event.ActionEvent evt) {
                    jButtonCancelActionPerformed(evt);
                }
            });
                    c.gridx=5;
                    c.gridy=0;
                    c.gridwidth=2;
                    panelDown.add(jButtonCancel,c);

                    c.gridx=9;
                    c.gridy=0;
                    panelDown.add(tm,c);

                container.add(panelDown,BorderLayout.SOUTH);

                setDefaultCloseOperation(DISPOSE_ON_CLOSE);
                this.pack();
                jComboBoxSchema.requestFocusInWindow();
                this.setMaximumSize(new Dimension(600,360));
                this.setSize(680,360);
                this.setVisible(true);
}

private void configureActionPerformed(ActionEvent e) {
        int[] i = jListTrajectoryTables.getSelectedIndices();
        if( i.length == 1){
                Object[] temp = jListTrajectoryTables.getSelectedValues();
                config.table = (String)temp[0];
      config.conn = conn;
      TrajectoryConfig tc = new TrajectoryConfig();
      tc.setConfig(config);
      tc.setVisible(true);
        }
        else{
                JOptionPane.showMessageDialog(this,"Select only one Trajectory Table.");
      return;
        }
}

private void jComboBoxMethodItemStateChanged(java.awt.event.ItemEvent evt) {
   Method alg = (Method) jComboBoxMethod.getSelectedItem();
   jComboBoxParam.removeAllItems();
   for (int i=0;i<alg.param.size();i++) {
      jComboBoxParam.addItem(alg.param.elementAt(i));
   }

   //prevents the SMoT methods to call upon parameters
   if(alg.toString().compareTo("SMoT")==0){
        jComboBoxParam.setEnabled(false);
        jTextFieldParam.setEnabled(false);
   }
   else{
        jComboBoxParam.setEnabled(true);
        jTextFieldParam.setEnabled(true);
   }
}

private boolean checkBufferState() {
   try{
```

```
            buffer = Double.valueOf(jTextFieldBuffer.getText());
            return true;

        } catch (NumberFormatException e){
            jTextFieldBuffer.setText("50.0");
            return false;
        }
    }

    private void jButtonGenArffFileActionPerformed(java.awt.event.ActionEvent evt) {
        GenArffFile gaf = new GenArffFile(conn,jRadioButtonFType.isSelected());
        gaf.setVisible(true);
    }

    private void jButtonCancelActionPerformed(java.awt.event.ActionEvent evt) {
        this.dispose();
    }

    @SuppressWarnings("static-access")
            private void jTextFieldParamFocusLost(java.awt.event.FocusEvent evt) {
        Parameter p = (Parameter) jComboBoxParam.getSelectedItem();
        try {
            if (p.type.DOUBLE == p.type) {
                p.value = new Double(jTextFieldParam.getText());
            }else {
                p.value = new Integer(jTextFieldParam.getText());
            }
        }catch(NumberFormatException e) {
            JOptionPane.showMessageDialog(this,"Parameter value invalid!");
        }
    }

    @SuppressWarnings("static-access")
            private void jComboBoxParamItemStateChanged(java.awt.event.ItemEvent evt) {
        Parameter p = (Parameter) evt.getItem();
        if (p.type == p.type.DOUBLE) {
            jTextFieldParam.setText(((Double)p.value).toString());
        }else {
            jTextFieldParam.setText(((Integer)p.value).toString());
        }
    }

    private void jComboBoxMethodItemStateChanged1(java.awt.event.ItemEvent evt) {
        Method alg = (Method) jComboBoxMethod.getSelectedItem();
        jComboBoxParam.removeAllItems();
        for (int i=0;i<alg.param.size();i++) {
            jComboBoxParam.addItem(alg.param.elementAt(i));
        }
    }

    private void jListRFValueChanged(javax.swing.event.ListSelectionEvent evt) {
        AssociatedParameter par = (AssociatedParameter) jListRF.getSelectedValue();
        if (par != null)
            RFMinTime.setText(""+par.value.intValue());
    }

    private void RFMinTimeFocusLost(java.awt.event.FocusEvent evt) {
        Object[] objs = jListRF.getSelectedValues();
        try {
            for (Object obj : objs) {
```

```
            AssociatedParameter p = (AssociatedParameter) obj;
            p.value = new Integer(Integer.parseInt(RFMinTime.getText()));
          }
          jTextFieldBuffer.grabFocus();
        }catch(java.lang.NumberFormatException e) {
          javax.swing.JOptionPane.showMessageDialog(this, e.toString());
          RFMinTime.grabFocus();
        }
    }

    private void RFMinTimeActionPerformed(java.awt.event.ActionEvent evt) {
        Object[] objs = jListRF.getSelectedValues();
        try {
          for (Object obj : objs) {
            AssociatedParameter p = (AssociatedParameter) obj;
            p.value = new Integer(Integer.parseInt(RFMinTime.getText()));
          }
          jTextFieldBuffer.grabFocus();
        }catch(java.lang.NumberFormatException e) {
          javax.swing.JOptionPane.showMessageDialog(this, e.toString());
          RFMinTime.grabFocus();
        }
    }


    private void LoadActionPerformed(ActionEvent evt) {
            try{ //load the tables in a list of auxiliary strings
          Statement s = conn.createStatement();
          ResultSet vTableName = s.executeQuery("SELECT f_table_name as tableName,type "+
                                    "FROM geometry_columns " +
                                    "WHERE                                    f_table_schema=trim('"+(String)
jComboBoxSchema.getSelectedItem()+"') "+
                                    "ORDER BY tableName");
          DefaultListModel model = (DefaultListModel) jListTrajectoryTables.getModel();//Trajct Tables
list
          model.removeAllElements();
          DefaultListModel model2 = (DefaultListModel) jListRF.getModel();//RF list
          model2.removeAllElements();
          while ( vTableName.next() )  {/* creates a new table for each table that has objects with
topological relation to vRegion */
            model2.addElement(new
AssociatedParameter(vTableName.getString("tableName"),vTableName.getString("type")));// RFs
            model.addElement(new String(vTableName.getString(1)));//Traject tables
          }
        }catch (Exception vErro){
          vErro.printStackTrace();
        }

    }

    private void OKActionPerformed(ActionEvent evt) throws IOException, SQLException {
            if(jCheckBoxBuffer.isSelected()){
                    if(checkBufferState()){
                            System.out.println("Buffer of "+buffer+" saved.");
                    }
                    else{
                            JOptionPane.showMessageDialog(this,"Buffer expects a number.");
                            return;
                    }
            } else {
```

```
                this.buffer = 50.0; // default value
            }

            if (jListRF.getSelectedIndex() == -1 &&
                    jComboBoxMethod.getSelectedItem().toString().compareTo("CB-SMoT")!=0){//cause
CB-SMoT has a version without RFs
             //cause DB-SMoT has a version without RFs too !
           JOptionPane.showMessageDialog(this,"Select one or more relevant features.");
           return;
        }

        if(jListTrajectoryTables.getSelectedIndex() == -1){
            JOptionPane.showMessageDialog(this,"Select one or more trajectory table.");
           return;
        }

        //get the thing in 'things', those trajectories tables to be executed
        Object[] objs = jListTrajectoryTables.getSelectedValues();
        String[] str = new String [objs.length];
        for(int i=0;i<objs.length;i++){
             str[i]=(String)objs[i];
        }

        //controls if SRID of RFs are different from trajectories...
                    // ALL the trajectories should have the SAME srid
                    // it is checked ahead in the foreach.
        config.table = str[0];
        String error = checkSRIDs();//att the variable 'table_srid'
        if(error.compareTo("")!=0){
            JOptionPane.showMessageDialog(this,error);
           return;
        }

        //for each of the trajectory table selected...
        for(int count =0; count<str.length;){
           java.util.Date tempo,fim,ini = new java.util.Date();
           config.table = str[count];
           try {
              //trajectory srid checking, has to be the same of all the other trajectory-talbes and RFs
              Statement sn = conn.createStatement();
              ResultSet rsn = sn.executeQuery("select srid from geometry_columns where
f_table_name='"+config.table+"'");
              rsn.next();
              if(table_srid!=rsn.getInt("srid")){
                 throw new Exception("SRID imcompatibles. Trajectory table "+config.table+" should be
changed.");
              }
              //enf of srid checking
              System.out.println("Creating tables...");
              createTables();
              System.out.println("Processing the trajectories...");

              loadTrajectories(config.table);
              fim = new java.util.Date();
              tempo = new java.util.Date(fim.getTime()-ini.getTime());
              //insertCleanTrajProcess(tempo.getTime());
              count++;
              if (count == str.length){
                 System.out.println("Processing time: " +tempo.getTime()+" ms");
                 JOptionPane.showMessageDialog(this,"Operation finished succesfully.");
```

```
            }
        }catch(Exception e) {
            e.printStackTrace();
            JOptionPane.showMessageDialog(this,"Error during operation");
            System.out.println("Error: \n"+e.getMessage());
        }
        finally{
        }
        //@Hercules, comented 30-08-2010.
//              if(objs.length>1){
//                  //if we have others tables of trajectories,
//                  //we should save the stops-table with another name to re-start the method
//                                  Statement st= conn.createStatement();
//                                  //save stops table
//                                  try{
//                                          st.execute("create table stops_"+s+" as (select * from
stops);");
//                                          System.out.println("stops_"+s+" created sucessfully.");
//                                  }
//                                  catch(SQLException e){
//                                          st.execute("create table stops_"+s+"_"+tempo.getTime()+" as
(select * from stops);");
//                                          System.out.println("stops_"+s+"_"+tempo.getTime()+"
created sucessfully.");
//                                  }
//              }
    }//endof foreach
    Runtime.getRuntime().gc();
  }

  private String checkSRIDs() {

    Statement sn;
                try {
                        //getting trajectory SRID
                        sn = conn.createStatement();
                        ResultSet rsn = sn.executeQuery("select srid from geometry_columns where
f_table_name='"+config.table+"'");
            rsn.next();
            table_srid=rsn.getInt("srid");

            //getting all the RFs
            Object[] objs = jListRF.getSelectedValues();
            AssociatedParameter[] relevantFeatures = new AssociatedParameter[objs.length];
            for (int i=0;i<objs.length;i++) {
              relevantFeatures[i] = (AssociatedParameter) objs[i];
            }
            //comparing their SRIDs with the trajectory
            for(AssociatedParameter a:relevantFeatures){
                sn = conn.createStatement();
                                rsn = sn.executeQuery("select srid from geometry_columns where
f_table_name='"+config.table+"'");
                        rsn.next();
                        if(table_srid!=rsn.getInt("srid")){
                            return "Error in the SRID of table: "+a.name;
                        }
            }
                } catch (SQLException e) {
                        System.out.println(e.getMessage());
                        e.printStackTrace();
```

```
                }
        return "";
        }

    //@autor: Bruno
        protected String showDadosGeograficos() {
                try {
                        //getting trajectory SRID

                        Statement s = conn.createStatement();

                System.out.println("++++++++++++++++++++");
                        if (jListTrajectoryTables.getSelectedIndex() != -1){
                                ResultSet vTableName = s.executeQuery("SELECT f_table_name as
tableName,type "+
                                                "FROM geometry_columns " +
                                                "WHERE f_table_schema=trim('"+(String)
jComboBoxSchema.getSelectedItem()+"') "+
                                                "ORDER BY tableName");
                                int indexAtualVT=0;
                                while ( vTableName.next() ) {

if(indexAtualVT==jListTrajectoryTables.getSelectedIndex()){

//System.out.println(vTableName.getString("tableName")+"-"+vTableName.getString("type"));

                                                ShowGeoData rep = new ShowGeoData(conn);

                                                JFrame f = new JFrame("Geographic Data
Visualizer");
                                                f.setFocusable(true);
                                                f.requestFocus();
                                                f.getContentPane().setLayout(new BorderLayout());
                                                f.getContentPane().add(rep,
BorderLayout.CENTER);
                                                f.pack();
                                                f.setVisible(true);
                                                f.setSize(new Dimension(800, 600));
                                                f.setResizable(false);

rep.loadPoints(vTableName.getString("tableName"),Color.BLUE,2);
                                                rep.paintAll(f.getGraphics());

//                                              // JUST FOR TESTING
//                                              f.addWindowListener(new  WindowAdapter(){public
void windowClosing(WindowEvent e) {
//                                                      System.exit(0);
//                                                      }});
//                                              // JUST FOR TESTING
                                        }
                                        indexAtualVT++;

                                }

                        }else{
                                System.out.println("Nada selecionado");
                        }

                System.out.println("---------------------");
```

```
                    } catch (SQLException e) {
                            System.out.println(e.getMessage());
                            e.printStackTrace();

                    }
                    return "";
            }
        //@autor: Bruno

//-----------------------------------------------------------------------------------------------------------
// INTERFACE ENDS
//-----------------------------------------------------------------------------------------------------------

/*
   public static void main (String args[]){
                    String url = "jdbc:postgresql://localhost:5432/";
                    String db = "";
                    String user = "";
                    String pass = "" ;
           new TrajectoryFrame(user,pass,url+db);
   }
*/



private void filterActionPerformed(ActionEvent e) {
   String esquema = (String)jComboBoxSchema.getSelectedItem();
   int[] i = jListTrajectoryTables.getSelectedIndices();
   if (i.length >= 1) {
      List<String> listaTab = new ArrayList<String>();
      Object[] temp = jListTrajectoryTables.getSelectedValues();
      for(Object ob: temp){
         listaTab.add((String) ob);
      }
      TrajectoryClean tc = new TrajectoryClean(conn, listaTab);
      tc.setConfig(this.config);
      tc.setEsquema(esquema);
      tc.setVisibleFrame(true);
   } else {
      JOptionPane.showMessageDialog(this, "Select one or more Trajectory Table.");
      return;
   }
}
public String formatNameParameter(String nm){
      System.out.println("Parm = "+nm);
      if(nm.trim().equalsIgnoreCase("MaxAvgSpeed")){
         return "as";
      }else if(nm.trim().equalsIgnoreCase("MinAvgSpeed")){
         return "as";
      }else if(nm.trim().equalsIgnoreCase("MinTime (seconds)")){
         return "mt";
      }else if(nm.trim().equalsIgnoreCase("MaxSpeed")){
         return "ms";
      }else if(nm.trim().equalsIgnoreCase("MinDirChange (degrees)")){
         return "md";
      }else if(nm.trim().equalsIgnoreCase("MaxTolerance (points)")){
         return "mt";
      }else if(nm.trim().equalsIgnoreCase("MinTimeVar (seconds)")){
         return "mtv";
      }else if(nm.trim().equalsIgnoreCase("MinTimeSpeed (seconds)")){
```

```
            return "mts";
        }
        return nm;
    }
    public String parametersClusterStr(){
        StringBuilder str = new StringBuilder();
        for(Parameter param :parametersCluster()){
            if(param.name!=null && !param.name.equals("") && param.value!=null){

str.append("_").append(formatNameParameter(param.name)).append("_").append(param.value.toString())
;
            }
        }
        String str1 = str.toString().toLowerCase().replace(".", "_")
                .replace(",", "_").replace(" ", "")
                .replace("(degrees)", "")
                .replace("(seconds)", "")
                .replace("(points)", "");
        return str1;
    }

    public List<Parameter> parametersCluster(){
        List<Parameter> list = new ArrayList<Parameter>();
        int tamParans = 0;
        while(tamParans < this.jComboBoxParam.getModel().getSize()){
            Parameter param = (Parameter)this.jComboBoxParam.getModel().getElementAt(tamParans);
            list.add(param);
            tamParans++;
        }
        return list;
    }

    public String nameTableStop(String sp){
        return "stops_".concat(sp.concat(parametersClusterStr()));
    }

    private static String currentNameTableStop;

    public static String getCurrentNameTableStop() {
        if(currentNameTableStop==null || currentNameTableStop.equals("")){
            return "stops";
        }
        return currentNameTableStop;
    }

    private static void setCurrentNameTableStop(String currentNameTableStop) {
        TrajectoryFrame.currentNameTableStop = currentNameTableStop;
    }

    public void insertCleanTrajProcess(long timeProcess){
        Object[] objs = jListRF.getSelectedValues();
        AssociatedParameter[] relevantFeatures = new AssociatedParameter[objs.length];
        Integer rfMinTime = null;
        if(relevantFeatures.length > 0){
            rfMinTime = relevantFeatures[0].value.intValue();
        }
        Double avgSpeed = null;
        Integer minTime = null;
        Double maxSpeed = null;
        int qtidadeStop = 0;
```

```
for(Parameter param: parametersCluster()){
   try{
      if(param.name.equalsIgnoreCase("MaxAvgSpeed")){
         avgSpeed = (Double)param.value;
      }else if(param.name.equalsIgnoreCase("MinAvgSpeed")){
         avgSpeed = (Double)param.value;
      }else if(param.name.equalsIgnoreCase("MinTime (seconds)")){
         minTime = (Integer)param.value;
      }else if(param.name.equalsIgnoreCase("MaxSpeed")){
         maxSpeed = (Double)param.value;
      }
   }catch(Exception e){
      e.printStackTrace();
   }
}

String sqlCount = "select count(*) from "+TrajectoryFrame.getCurrentNameTableStop();
System.out.println(sqlCount.toString());
try{
   PreparedStatement psCount = conn.prepareStatement(sqlCount);
   ResultSet resultSet = psCount.executeQuery();
   if(resultSet.next()){
      qtidadeStop = resultSet.getInt(1);
   }
}catch(SQLException sqle){
   sqle.printStackTrace();
}

StringBuilder sql = new StringBuilder();
   sql.append(" INSERT INTO cleantrajprocess( ");
   sql.append("     oidstop, nomestop, tempostop, paramrfmintime, paramminavgspeed,  ");
   sql.append(" parammintime, parammaxspeed, qtidadestop) ");
   sql.append(" VALUES (null, ?, ?, ");
   sql.append(rfMinTime==null?" null,":" ?, ");
   sql.append(avgSpeed ==null?" null,":" ?, ");
   sql.append(minTime  ==null?" null,":" ?, ");
   sql.append(maxSpeed ==null?" null,":" ?, ");
   sql.append(" ? ) ");


try {
   System.out.println(sql.toString());
   int i=1;
   PreparedStatement ps = conn.prepareStatement(sql.toString());
   ps.setString(i++, TrajectoryFrame.getCurrentNameTableStop());
   ps.setLong  (i++, timeProcess);
   if(rfMinTime!=null){
      ps.setInt   (i++, rfMinTime);
   }
   if(avgSpeed!=null){
      ps.setDouble(i++, avgSpeed);
   }
   if(minTime!=null){
      ps.setInt   (i++, minTime);
   }
   if(maxSpeed!=null){
      ps.setDouble(i++  , maxSpeed);
   }
   ps.setInt(i, qtidadeStop);
```

```
            ps.execute();
        } catch (SQLException ex) {
            Logger.getLogger(TrajectoryFrame.class.getName()).log(Level.SEVERE, null, ex);
        }

    }

public static void main(String args[]) {
    String url = "jdbc:postgresql://localhost:5432/";
    String db = "postgis";
    String user = "postgres";
    String pass = "postgres";
    new TrajectoryFrame(user, pass, url + db);
}


}
```