ANPET
XXVIII Congresso de Pesquisa e Ensino em Transportes
CURITIBA — PR

UMA FERRAMENTA PARA ALOCAÇÃO DE TRÁFEGO E APRENDIZAGEM DE ROTAS EM REDES VIÁRIAS

Ricardo Grunitzki

Gabriel de Oliveira Ramos

Ana Lucia Cetertich Bazzan

Agência Brasileira do ISBN
ISBN 978-85-87893-17-8
9 788587 893178

anpet

UMA FERRAMENTA PARA ALOCAÇÃO DE TRÁFEGO E APRENDIZAGEM DE ROTAS EM REDES VIÁRIAS

Ricardo Grunitzki
Gabriel de Oliveira Ramos
Ana L. C. Bazzan
Universidade Federal do Rio Grande do Sul
Instituto de Informática

RESUMO
Este trabalho apresenta uma ferramenta desenvolvida em Java para realizar alocação de tráfego e aprendizagem de rotas em redes viárias. Para alocação de tráfego, a ferramenta disponibiliza os métodos: *all-or-nothing assignment*, *incremental assignment* e *successive averages*. Já para aprendizagem de rotas, os métodos *Q-learning* e *learning automata* são disponibilizados. Além destes métodos, a ferramenta apresenta uma estrutura que permite que novos métodos sejam facilmente implementados. Este artigo descreve a arquitetura e implementação da ferramenta, bem como a sua utilização. Uma análise do custo médio de viagem, obtido em um cenário abstrato, é realizada para demonstrar a eficiência dos diferentes métodos implementados.

ABSTRACT
This paper presents a tool developed in Java to perform traffic assignment and route learning in road networks. For traffic assignment, the tool provides the following methods: all-or-nothing assignment, incremental assignment, and successive averages. As for route learning, Q-learning and learning automata methods are available. In addition to these methods, the tool presents a structure that allows new methods to be easily implemented. This paper describes the architecture and implementation of the tool, as well as their usage. An analysis of the average travel cost, obtained in an abstract scenario, is performed to demonstrate the effectiveness of different methods.

1. INTRODUÇÃO

Realizar a conexão entre oferta e demanda é um tema recorrente na literatura e uma tarefa muito importante durante a modelagem e simulação de um sistema de transporte. Para realizar esta tarefa, métodos de alocação de tráfego (AT) e aprendizagem de rotas (AR) tem sido propostos. Eles são responsáveis por conectar a infraestrutura física (oferta) à demanda (viagens, veículos ou motoristas) que irá utilizá-la. Para Ortúzar e Willumsen (2001, Captítulo 1), a partir desta conexão é possível, por exemplo, avaliar o impacto de alterações na oferta ou demanda, no sistema de transporte.

Assim como na AT, onde há uma constante busca por métodos cada vez mais eficientes, realizar a conexão entre oferta e demanda se torna um cenário desafiador do ponto de vista de abordagens de aprendizado por reforço multiagente. No aprendizado por reforço multiagente (Buşoniu *et al.*, 2008), cada viagem é considerada um agente capaz de aprender individualmente qual a melhor rota que satisfaz suas restrições de origem e destino. Portanto, o agente precisa aprender uma rota na presença de outros agentes também capazes de aprender. Embora a literatura não apresente uma terminologia única para endereçar tais abordagens, o presente trabalho utiliza o termo "aprendizagem de rotas" (AR). A principal diferença entre AT e AR está na descentralização da tomada de decisão. Na AT, existe uma autoridade central responsável por determinar a rota de cada viagem. Já no AR, por sua vez, não existe uma entidade central, cada viagem é responsável por definir qual rota deve utilizar.

A literatura apresenta uma série de ferramentas de simulação microscópica (EMME, TransModeler e etc.) que contêm módulos para realizar a conexão entre oferta e demanda - em geral, métodos de AT. Porém, por se tratar de softwares proprietários, o uso é limitado, sobretudo no

meio acadêmico. Por outro lado, algumas ferramentas como o SUMO e ITSUMO são apresentadas para suprir esta necessidade. Entretanto, como estas ferramentas simulam o sistema de transporte a nível microscópico, isto implica em um aumento considerável de detalhes sobre a simulação. Como consequência, há um maior número de parâmetros do cenário a serem definidos e um custo computacional mais elevado.

O presente trabalho apresenta uma ferramenta para realizar AT e AR em redes viárias. A ferramenta dispõe de uma estrutura flexível que permite utilizar os métodos existentes ou desenvolver novos, de modo simples. Os métodos que acompanham a ferramenta são classificados em duas categorias: AT e AR. Os métodos de AT implementados são: *all-or-nothing*, *incremental-assignment* e *successive averages*. Já os métodos de AR são: Q-*learning* com estados, Q-*learning* sem estados e *learning automata*. Para que um cenário qualquer seja avaliado na ferramenta - tanto por métodos de AT quanto AR -, basta que a rede viária, demanda e função de custo estejam definidas de acordo com os formatos aceitos pela ferramenta. Experimentos são realizados em um cenários abstrato para demonstrar os resultados obtidos pelos métodos implementados na ferramenta. Na avaliação do custo de viagem, os métodos de AR apresentaram resultados superiores aos métodos de AT.

O trabalho está organizado da seguinte forma. A Seção 2 apresenta os fundamentos de oferta e demanda em sistemas de transporte, AT e AR bem como os métodos implementados na ferramenta. A Seção 3 apresenta as principais características da ferramenta, a arquitetura utilizada no seu desenvolvimento, as etapas do processo de execução e sua utilização. Resultados experimentais são apresentados na Seção 4. A Seção 5 conclui este trabalho e apresenta perspectivas de continuação.

2. OFERTA E DEMANDA EM SISTEMAS DE TRANSPORTE

Um sistema de transporte é visto por Ortúzar e Willumsen (2001, Capítulo 1) como um composto de uma parte de oferta (estrutura física) e outra parte chamada demanda (viagens). A representação da oferta, em termos de infraestrutura de transporte, pode ser feita através dos conceitos de teoria dos grafos. Isto posto, a rede é representada por um grafo $G\left(V, L\right)$, onde conjunto de vértices V representa os cruzamentos ou intersecções e o conjunto de links L representa as vias que ligam os cruzamentos. Cada link $l_k \in L$ possui um custo c_k - variável de acordo com a função de custo -, o qual é representado por alguma forma de custo associada à travessia do mesmo, como: seu comprimento, tempo de viagem para trafegá-lo, quantidade de combustível consumido, entre outros. A demanda, por outro lado, representa o desejo de determinadas entidades em se locomoverem de um lugar para outro. Dentre as formas de modelagem de demanda existentes, Arfaoui (1999) afirma que é possível dividir a rede em zonas, distritos ou centroides (a partir de dados socioeconômicos, densidade populacional, entre outros fatores). Por meio destes dados, uma matriz T chamada de matriz origem-destino (matriz OD) é construída. Esta matriz contém I linhas (zonas de origem) e J colunas (zonas de destino). Cada elemento T_{ij} representa o número de viagens da zona i para a zona j, em um dado intervalo de tempo. Diz-se que $i \in I$ e $j \in J$ são um par origem-destino (ou par OD).

Para que seja possível determinar o fluxo de uma rede de tráfego, deve-se determinar quais partes da infraestrutura as viagens estão dispostas a utilizar. Isto significa que, de alguma forma, a demanda precisa ser alocada à oferta disponível. A conexão entre origem i e destino j geralmente não consiste em apenas um único link, mas uma sequência de vértices conectados

$(v_0, v_1, v_2, \ldots)$ que juntos formam uma rota r_p. Dentre as diversas formas de avaliação do custo por trafegar em um link, pode-se destacar o tempo de viagem para percorrê-lo. Este tempo de viagem pode ser representado por uma VDF (em inglês: *volume-delay function*). Esta função expressa o tempo médio de viagem no link de acordo com o seu volume de tráfego. Um exemplo de VDF é a sugerida pelo BPR (Bureau of Public Roads): $t_l = t_{l_0}\left[1 + \mathcal{A}_l\left(\frac{V_l}{C_l}\right)^{\mathcal{B}_l}\right]$. Nesta função, t_{l_0} representa o tempo de viagem sob a condição de fluxo livre de congestionamento no link l, V_l é o volume (em veículos por unidade de tempo) e C_l é a capacidade do link. $\mathcal{A}_l$ e $\mathcal{B}_l$ são fatores que devem ser empiricamente definidos para cada link, para determinar como o tempo de viagem deve reagir ao aumento do volume.

As próximas seções apresentam uma breve descrição sobre as técnicas de AT e AR, bem como seus métodos implementados na ferramenta.

2.1. Alocação de tráfego

De modo geral, AT seleciona rotas que conectam as origens das viagens com seus respectivos destinos presentes na matriz OD (Ortúzar e Willumsen, 2001, Capítulo 10). A literatura apresenta uma série de técnicas para realizar esta tarefa, desde métodos determinísticos e clássicos até métodos de computação evolutiva mais sofisticados. A grande similaridade dos métodos é o processo de seleção de rotas, o qual é realizado de forma centralizada. Em outras palavras, considera-se que uma entidade central determina as rotas que cada viagem deverá utilizar. As seções subsequentes apresentam os métodos de AT disponíveis na ferramenta.

2.1.1. All-or-nothing assignment

O *All-or-nothing assignment*, apresentado por Ortúzar e Willumsen (2001, Capítulo 10), é o método mais simples existente de AT. Ele assume que não existem efeitos de congestionamentos na rede, tal que para todas as viagens são considerados os mesmos atributos da rede durante o cálculo da rota. A ausência dos efeitos dos congestionamentos implica diretamente na escolha de rota. A AT deste método é dada pela Equação 1. Como pode ser observado, todas as viagens com origem em i e destino em j utilizarão a mesma rota de menor custo r^*. Para Ortúzar e Willumsen (2001, Capítulo 10), esta suposição pode ser razoável em redes esparsas ou onde há poucas rotas alternativas e muita diferença de custo entre elas.

$$\left.\begin{array}{ll} T_{ijr^*} & = T_{ij} \quad \text{para a rota de menor custo } r^* \\ T_{ijr} & = 0 \quad \text{para todas as demais rotas} \end{array}\right\} \forall_{i,j} \tag{1}$$

2.1.2. Incremental assignment

O *Incremental Assignment*, apresentado no Capítulo 10 de Ortúzar e Willumsen (2001), é uma abordagem um pouco mais realista quando comparada ao *all-or-nothing assignment*. Neste método, divide-se o total de viagens da matriz T (para cada par OD) em n frações de matriz, de acordo com um fator de proporção p_n, tal que $\sum_n p_n = 1$. As matrizes fracionadas são carregadas na rede de forma incremental (por exemplo, 10%, 20% e etc.). Após cada fração de matriz ser carregada, os pesos dos links são novamente calculados. Segundo Ortúzar e Willumsen (2001, Capítulo 10), são valores típicos de p_n: 0.4, 0.3, 0.2 e 0.1. Uma grande desvantagem deste método é que após um fluxo ser atribuído a um link, este fluxo não será mais removido e alocado para outro link. Logo, se as iterações iniciais alocarem muitas viagens em um determinado link, provavelmente o método não convergirá para uma boa solução. Apesar disso, os

autores ressaltam que este método proporciona duas principais vantagens: facilidade de programação e resultados de cada iteração interpretáveis como os congestionamentos formados em períodos de pico.

2.1.3. Successive averages

O método *successive averages* também é um método iterativo de AT, porém mais eficiente quando comparado com ao *incremental* e *all-or-nothing assignment*, pois a cada iteração as rotas são recalculadas de acordo com os novos custos dos links (Ortúzar e Willumsen, 2001, Capítulo 10). Este método aloca rotas às viagens a partir de repetidas execuções do *all-or-nothing*. A cada repetição, custo e volume dos links são atualizados por meio da distribuição das viagens nas iterações anteriores. Este processo é repetido até que não haja uma mudança significativa no custo dos links ou volumes. Por exemplo, no método *successive averages*, o fluxo na iteração n é calculado como uma combinação linear do fluxo na iteração $n-1$ e de um fluxo auxiliar resultante da execução do *all-or-nothing* na iteração n.

2.2. Aprendizagem de rotas

Através do aprendizado por reforço multiagente é possível realizar a conexão entre demanda e oferta, de forma inteligente e distribuída. Esta técnica é muito utilizada onde uma política de escolha de ações - rota, no caso de AR - deve ser aprendida (Buşoniu *et al.*, 2008). No contexto de AR, as viagens da demanda são representadas por agentes capazes de aprender por reforço, e a rede de tráfego é modelada como o ambiente dos agentes. A tarefa de aprendizagem dos agentes é encontrar a rota com custo mais atraente, que satisfaça suas restrições de origem e destino. Apesar de esta tarefa ser usualmente referida na literatura como "escolha de rotas", este trabalho utiliza a denominação "aprendizagem de rotas". A motivação para esta distinção é que, dependendo do método de AR, os agentes podem construir a rota que conecta sua origem e destino ao longo da viagem, ou selecionar uma rota completa dentre as disponíveis em um conjunto de pré-definido.

2.2.1. Q-learning

O algoritmo *Q-learning* (Watkins e Dayan, 1992) é um dos mais conhecidos algoritmos de aprendizado por reforço. Para a tarefa de AR, podemos caracterizar a rota como a tarefa de aprendizagem. As rotas são aprendidas de forma individual por cada motorista através de sucessivas interações com o ambiente. Esta ferramenta apresenta duas variantes do *Q-learning* para AR: *Q-learning* com estados e *Q-learning* sem estados. Na abordagem sem estados, os agentes aprendem a rota a partir de um conjunto de rotas pré-definidas K. Já na abordagem com estados, a rota é construída ao longo da viagem. A diferença entre as duas abordagens está na complexidade da tarefa de aprendizagem. Considerando que os motoristas podem andar em círculos na abordagem com estados, o número de rotas entre um par OD pode ser infinito. Por outro lado, esta abordagem não limita a tarefa de aprendizagem a um número pré-definido de soluções. O algoritmo *Q-learning* possui três parâmetros: uma taxa de aprendizagem α, um fator de desconto γ e uma taxa de exploração ϵ. Uma descrição mais abrangente sobre o *Q-learning* com estados para a tarefa de AR pode ser encontrada em Grunitzki *et al.* (2014) e para o *Q-learning* sem estados em Tumer e Agogino (2006).

2.2.2. Learning automata

O *learning automata* (Narendra e Thathachar, 1989) é um método de aprendizagem por reforço onde a política é representada na forma de um vetor de probabilidades. As ações são escolhidas

aleatoriamente com base no referido vetor de probabilidades. A atualização da política, por sua vez, se dá através de um mecanismo chamado *Linear Reward–Inaction*, que atualiza proporcionalmente as probabilidades das ações com base nas recompensas obtidas. A vantagem deste mecanismo probabilístico de seleção das ações é que ele nunca anula a exploração de ações que possuem recompensa menor. Além do mais, este mecanismo costuma aprender de maneira mais rápida do que outros algoritmos. No que se refere a parâmetros, o *learning automata* original possui apenas uma taxa de aprendizagem, representada por α. A versão do *learning automata* incluída nesta ferramenta, no entanto, é uma modificação proposta por Ramos *et al.* (2014). Nesta, cada par OD possui um conjunto K de rotas de menor custo. Além do mais, o conjunto de rotas é atualizado com probabilidade ω. Através desta modelagem, busca-se promover uma maior diversidade de rotas e consequentemente distribuir melhor o fluxo de veículos na rede viária.

3. FERRAMENTA PROPOSTA

As seções subsequentes apresentam a arquitetura, técnicas utilizadas e uma breve demonstração de utilização da ferramenta proposta.

3.1. Arquitetura e implementação

A ferramenta foi inteiramente desenvolvida em linguagem Java e seu código-fonte está organizado em camadas. Como pode-se observar na Figura 1, o código é dividido em seis pacotes de classe. O pacote *simulation* contém as classes responsáveis por realizar a leitura dos arquivos de rede e demanda (*ProcessFiles*), controlar a execução dos métodos (*Simulation*) bem como a predefinição de parâmetros (*Params*).

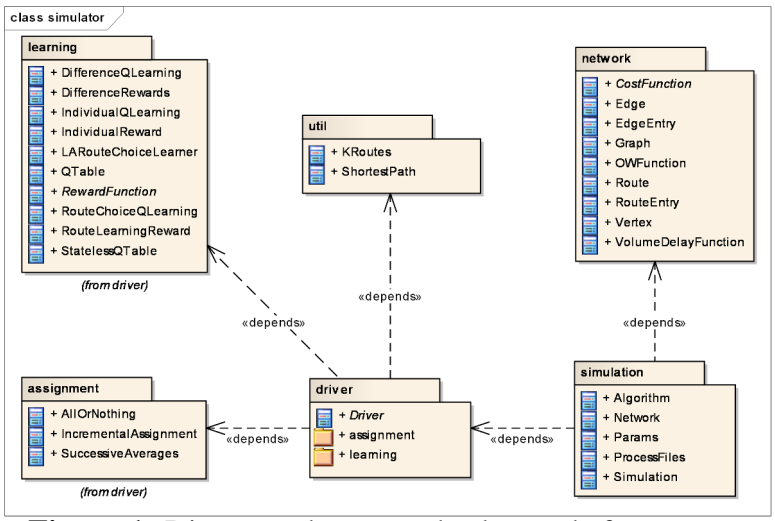

Figura 1: Diagrama de pacote de classes da ferramenta.

O pacote *driver* apresenta a classe básica de motorista utilizado. Independente do método ser de AT ou AR, nesta ferramenta, cada viagem da matriz OD é representada por um motorista do tipo *Driver*. Logo, para cada viagem da matriz OD, deve existir um objeto *Driver* que modele o seu comportamento. A classe *Driver* implementa a estrutura básica utilizada para modelar o comportamento dos motoristas bem como seus atributos. Os algoritmos dos pacotes *learning* e *assignment* estendem a classe *Driver* e modelam a lógica do método de acordo com as suas características.

As classes do pacote *network* definem a estrutura básica que representa a rede viária. A rede é representada por uma grafo do tipo *Graph*. Este grafo possui um conjunto de links representados por objetos do tipo *Edge* e um conjunto de vértices do tipo *Vertex*. Um objeto *Edge* possui todos os atributos que definem um link, e uma função de custo *CostFunction*. Um objeto do tipo *Vertex* representa um cruzamento da rede viária, o qual possui um conjunto de links que o conectam aos demais vértices. A classe *CostFunction* é uma classe abstrata que contém a assinatura do método responsável por avaliar a função de custo. As classes *VolumeDelayFunction* e *OWFunction* são implementações das funções de custo utilizadas nos cenários disponíveis junto a ferramenta.

No pacote *util* são apresentadas algumas classes utilitárias, utilizadas por alguns dos métodos já implementados. A classe *KRoutes* possui as rotinas responsáveis por encontrar as K rotas de menor custo em um *Graph*. Já a classe *ShortestPath* é a implementação do algoritmo Dijskstra para encontrar o menor caminho em um objeto *Graph*.

3.2. Etapas do processo de execução da ferramenta

A Figura 2 apresenta de forma abstrata as etapas do processo de execução da ferramenta. Na primeira etapa, todos os parâmetros que envolvem a execução, o método utilizado e o cenário devem ser definidos. O conjunto de parâmetros é variável em função do método utilizado. Uma descrição completa dos parâmetros utilizados é apresentada junto à documentação do código-fonte de cada método. A Seção 3.3 apresenta um exemplo para facilitar o entendimento destes parâmetros.

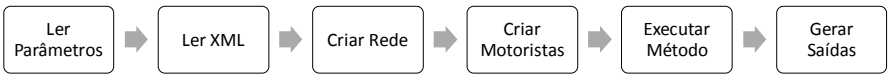

Figura 2: Etapas do processo de execução.

Na segunda etapa, são lidos os dois arquivos XML que definem a rede viária e matriz OD. Estes arquivos são gerados a partir do cenário a ser avaliado. A Figura 3 apresenta a conversão de um exemplo hipotético para o formato reconhecido pela ferramenta. O primeiro arquivo, com a extensão *.net.xml*, representa a rede viária. No exemplo, este arquivo possui em seu elemento raiz <*graph*> dois elementos filhos: <*nodes*> e <*edges*>. Os elementos <*nodes*> representam os vértices da rede e possuem apenas um atributo identificador que deve ser único. Já os elementos <*edges*> representam os links da rede. Os *edges* possuem um atributo identificador único (*id*), um vértice origem (*from*), um vértice destino (*to*), um custo em fluxo livre (*ffttime*), capacidade (*capacity*), dois parâmetros empiricamente definidos para a função de custo desejada (*alpha* e *beta*) e um identificador (*directed*) se o link é direcionado ou não. Este conjunto básico de parâmetros foi definido com base na VDF apresentada na Seção 2.1, mas a estrutura da ferramenta é flexível para aceitar novos parâmetros.

O arquivo com extensão *.od.xml* representa a matriz OD do cenário. O arquivo possui apenas um elemento filho em seu elemento raiz <*od-matrix*>: o elemento <*od*>, que representa os pares OD da matriz. Cada elemento possui um atributo origem (*origin*), um atributo destino (*destination*) e um atributo *trips* com o número de viagens do respectivo par OD. Esta ferramenta considera que as zonas de origem e destino da matriz OD são vértices definidos no arquivo da rede viária.

Após a leitura dos arquivos XML na segunda etapa, caso os arquivos sejam validados pela fer-

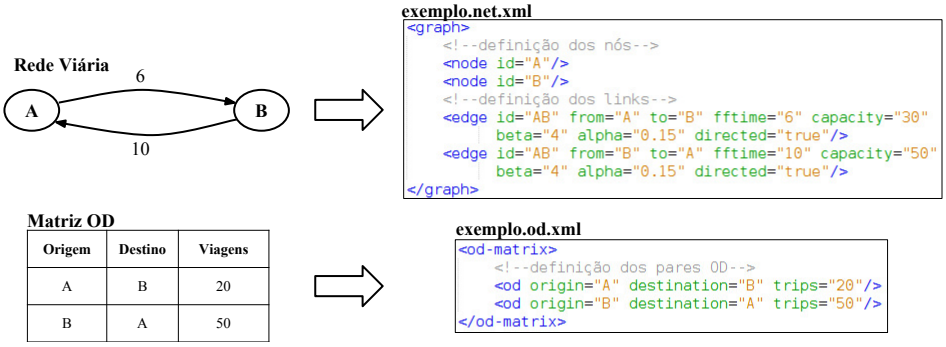

Figura 3: Conversão da rede viária e matriz OD de um problema hipotético para o formato XML da ferramenta.

ramenta e nenhuma mensagem de erro seja disparada, são criados a rede viária (terceira etapa) e os motoristas (quarta etapa). A rede viária é um grafo criado com base no arquivo *.net.xml*. Os links da rede possuem os atributos definidos no arquivo *.net.xml* e atributos referentes à execução da ferramenta como: quantidade atual de veículos, fluxo, volume, função de custo e custo de viagem. Os motoristas são criados com base no método definido na primeira etapa. Lembrando que cada viagem da matriz OD representa um motorista nesta ferramenta.

Na quinta etapa da Figura 2, é realizada a execução do método. Para tornar possível a implementação de vários métodos foi adotado o seguinte padrão. Uma execução completa, onde todos os veículos se deslocam de sua origem até o seu destino, é chamada de episódio. Em função de existirem métodos iterativos como o *incremental assignment, successive averages*, *Q-learning* com e sem estados e *learning automata*, a ferramenta permite que sejam executados vários episódios em uma única execução da ferramenta. Cada transição de links dentro de um episódio representa um passo de tempo. Desta forma, para um motorista percorrer uma rota com 10 links, ele levará 10 passos de tempo. Tanto o número máximo de passos de tempo quanto o de episódios são parâmetros.

Com base nas definições de passos de tempo e episódios, o processo de execução do método é realizado de acordo com o fluxograma da Figura 4. Neste fluxograma, o processo de execução do método *all-or-nothing assignment* é apresentado. Nota-se que as operações executadas pelos motoristas *all-or-nothing*, realizadas simultaneamente por todos, são as implementações dos comportamentos específicos do método para cada uma das etapas deste processo. A execução do método segue o padrão (processamento inicial, passo A, passo B e processamento final) para tornar a implementação genérica e estendível para qualquer método. Neste exemplo, utilizamos o método *all-or-nothing* por ser o método mais simples entre os implementados na ferramenta. Vale ressaltar que, por tratar-se de um método não iterativo, esta execução possui apenas um episódio. Para o número máximo de passos, é definido um valor grande o bastante para que todos os veículos cheguem ao seu destino. Assim, o método é executado da seguinte maneira. No *processamento inicial* do episódio, todos os motoristas inicializam seus atributos (como custo de viagem, posição atual e assim por diante) e calculam a rota de menor custo a ser percorrida (a partir do algoritmo *Dijskstra*). Após isso, na implementação do *passo A*, todos os motoristas selecionam o próximo link da rota anteriormente calculada e o percorrem. Esta ação dos motoristas irá gerar um fluxo na rede. De posse deste fluxo, a operação *atualiza rede* atualiza o custo links. Em seguida, com base nos novos custos, os motoristas incrementam

no *Passo B* a rota percorrida. Este processo é repetido até que não hajam mais veículos na rede viária ou que o número máximo de passos seja atingindo. Após os passos, com base nos custos de viagem da rota percorrida, é realizado o cálculo do custo total de viagem. Ao final do episódio, é executada a sexta etapa da Figura 2, onde são geradas as saídas desejadas (ver Seção 3.3). As saídas podem ser impressas no terminal ou gravadas em arquivo texto. Ao fim deste processo, a execução da ferramenta é concluída.

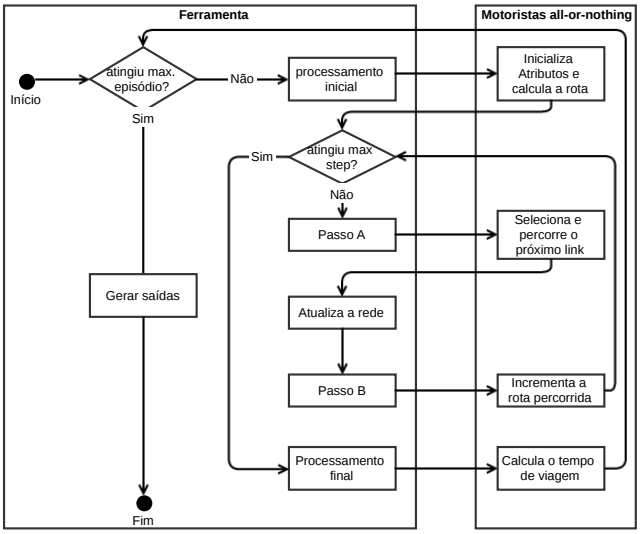

Figura 4: Fluxograma do processo de execução do método *all-or-nothing assignment*.

Vale destacar algumas características deste processo que tornam a execução dos métodos mais eficiente. O modelo básico de motorista (*Driver*, apresentado na Seção 3.1) já possui algumas rotinas paralelizadas através das operações *Passo A* e *Passo B*, as quais tornam o processo de execução dos métodos muito mais eficiente. Desta forma, é possível que os motoristas sejam processados de forma paralela, gerando um ganho de desempenho. Por padrão, a ferramenta distribui o processamento entre os núcleos de processamento disponíveis na máquina em que está sendo executada. Este número também é parametrizável.

3.3. Utilização da ferramenta

Esta seção apresenta um exemplo de utilização da ferramenta. Utilizar a ferramenta requer conhecimentos básicos da linguagem de programação Java. Outro pré-requisito é possuir o JDK 1.7 ou superior instalado no computador em que a ferramenta será executada. Em seguida, o próximo passo é adquirir o código-fonte da ferramenta através do seguinte repositório: `https://code.google.com/p/difference-on-routing`. A ferramenta não necessita de bibliotecas adicionais.

A Figura 5 apresenta um exemplo de AT utilizando o algoritmo *all-or-nothing* no cenário OW (ver Seção 4.1). Além deste cenário, o diretório *\files* apresenta também o cenário *Sioux Falls* (disponível em: `http://www.bgu.ac.il/~bargera/tntp/`) no formato XML. Entre as linhas 2 e 6 são definidos os parâmetros da execução. A linha 5 representa um parâmetro opcional utilizado para multiplicar a quantidade de viagens dos pares OD por um determinado fator. Por padrão este valor é sempre 1. A linha 6 indica que os resultados da execução serão impressos no terminal. A classe *simulation.Params* apresenta a lista completa de parâmetros com sua respectiva descrição de utilização. Instruções sobre como gravar os resultados em

arquivo texto estão disponíveis na documentação da classe.

```
1  /* Parâmetros */
2  Params.OD_MATRIZ_FILE = "files/ow.od.xml";//caminho da matriz OD
3  Params.NETWORK_FILE = "files/ow.net.xml";//caminho da rede
4  Params.NUM_EPISODES = 1;//número máximo de episódios
5  Params.DEMAND_FACTOR = 1;//fator de multiplicação da demanda
6  Params.PRINT_ON_TERMINAL = true;//imprime resultados no terminal
7  /*Gera a rede*/
8  File net_xml = new File(Params.NETWORK_FILE);//arquivo com a rede
9  CostFunction function = new OWFunction(); //função de custo
10 Graph network = ProcessFiles.processGraph(net_xml, function); //rede
11 /*Gera a demanda*/
12 File od_xml = new File(Params.OD_MATRIZ_FILE);//arquivo com a matriz OD
13 List<Driver> drivers = ProcessFiles.processODMatrix(network, od_xml,
14 Params.DEMAND_FACTOR, AllOrNothing.class);//lista de motoristas
15 /*Gera e inicia uma execução do método*/
16 Simulation simulation = new Simulation(network, drivers);//controla a execução do método
17 simulation.start();//executa o método
18 /*Gera os resultados*/
19 simulation.printCostResults();//imprime os custos médios
20 //simulation.printLinksFlow();imprime os fluxos dos links da rede
```

Figura 5: Exemplo de execução do método *all-or-nothing* na ferramenta.

Após a definição dos parâmetros é necessário criar uma rede viária. Na linha 10 é possível perceber que a criação da rede exige dois objetos, uma função de custo do tipo *CostFunction* (linha 9) e um objeto do tipo *File* com o arquivo *.net.xml* (linha 8). A função de custo recebe uma instância da classe *OWFunction* por ser a implementação da função de custo do cenário em questão. O objeto do tipo *Graph* armazena o grafo da rede viária gerada.

O próximo passo é a criação da demanda. A demanda é representa por uma lista de objetos do tipo *Driver* (linha 13). O método de criação da demanda recebe quatro parâmetros. Uma rede viária (*network*), um objeto com o arquivo XML da matriz OD (*od_xml*), um fator de multiplicação da demanda (*Params.DEMAND_FACTOR*) e o tipo *Class* do método a ser utilizado (*AllOrNothing.class*). Para alterar o método basta alterar o último parâmetro pelo desejável. O arquivo *Main.java*, que acompanha o código fonte, apresenta exemplos de utilização para cada um dos algoritmos disponíveis na ferramenta.

Com a demanda (*drivers*) e rede viária (*network*) devidamente criados, basta instanciar um objeto do tipo *Simulation* (linha 16). Este objeto é responsável por realizar a execução do método. A linha 17 executa o método. Após a execução do método, a linha 19 irá gerar os custos de viagem. O resultado final é o apresentado na Figura 6. A primeira linha representa o cabeçalho dos resultados. As colunas são separadas por ";" e as quebras de linha por "\n". A primeira coluna representa o episódio (por se tratar de um método não iterativo, nesse caso, só existe um episódio), a segunda representa o custo médio de viagem na rede (avg) e as demais são os custos médios de viagem por par OD. Além do custo de viagem, a ferramenta também pode gerar outros resultados. Por exemplo, o comando da linha 20 na Figura 5, retornaria os volumes de veículos por link da rede, caso não estivesse comentado. Seguindo a documentação do código também é possível criar novos métodos para compilação dos resultados.

```
episode;avg;A-L;A-M;B-L;B-M
1;88.8235;114.0000;78.0000;98.0000;55.0000
```

Figura 6: Exemplo de saída da ferramenta para a métrica custo de viagem.

4. EXPERIMENTOS

No intuito de demonstrar a eficiência dos diferentes métodos disponíveis na ferramenta, nesta seção é realizada uma comparação entre os resultados obtidos por eles. As seções a seguir apresentam o cenário utilizado, a escolha de parâmetros dos métodos e os resultados obtidos em termos de tempo de viagem.

4.1. Cenário

Este cenário é uma adaptação do problema apresentado em Ortúzar e Willumsen (2001, Exercício 10.1). A topologia da rede de tráfego é ilustrada na Figura 7 e possui 13 vértices e 48 links. O valor sobre cada link representa o tempo de viagem sem congestionamento (t_{l_0}) para ambos os sentidos. A rede possui duas zonas de origens (A e B), representando duas áreas residenciais, e duas zonas de destino (L e M), representando dois shoppings centers.

Este cenário, apesar de parecer simples, apresenta uma série de características relevantes para a análise de desempenho dos métodos. Por exemplo, o cenário captura algumas propriedades importantes do mundo real como o compartilhamento de links entre rotas distintas. Apesar disso, para tornar o cenário mais complexo, modificações nos custos de alguns links foram realizadas. Os pesos dos links AC, BE, CF, EH e GJ (e sentidos opostos) foram significativamente aumentados para torná-los menos atrativos (de fato, a Figura 7 apresenta a versão modificada da rede). Esta versão da rede também está disponível no repositório da ferramenta.

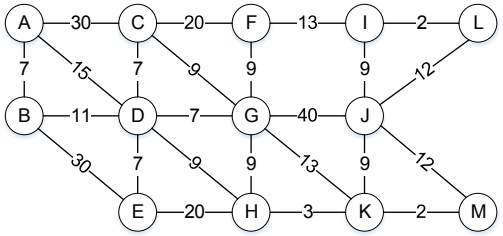

Figura 7: Topologia da rede OW adaptada de Ortúzar e Willumsen (2001, Exercício 10.1). Os links representam os dois sentidos da via.

Assim como em Ortúzar e Willumsen (2001, Exercício 10.1), a matriz OD utilizada no presente trabalho possui 1700 viagens distribuídas em 4 pares OD, como mostra a Tabela 1. A função de custo proposta pelos autores representa o tempo de viagem no link em minutos e é definida por $t_l = t_{l_0} + 0.02 * q_l$, onde, t_l é o tempo de viagem no link l, t_{l_0} é o tempo de viagem sem congestionamento e q_l é o volume do link. Nesta função, o tempo de viagem é incrementado em 0.02 por veículo.

Tabela 1: Pares OD da rede OW (Ortúzar e Willumsen, 2001, Exercício 10.1).

origem	destino	viagens
A	L	600
A	M	400
B	L	300
B	M	400
	total	1700

4.2. Parâmetros dos métodos

Os parâmetros utilizados pelos métodos *incremental assignment* e *successive averages* são definidos com base no trabalho de Ortúzar e Willumsen (2001, Exercício 10.1). Para o mé-

todo *successive averages* utilizou-se $\phi = \frac{1}{n}$, 100 iterações como critério de parada (equivalente aos episódios) e 100 passos de tempo. Para o *incremental assignment* utilizou-se $p_n = \{0.4, 0.3, 0.2, 0.1\}$. O método *all-or-nothing* não possui parâmetros.

Para o Q-*learning* sem estados os parâmetros foram definidos empiricamente: $\alpha = 0.05$, $\gamma = 0.99$, $\epsilon = 1.0$ com uma taxa de decaimento de 0.915 por episódio, 100 passos de tempo e 50 episódios. Para os demais métodos baseados em aprendizado por reforço, utilizamos o mesmo conjunto de parâmetros de trabalhos que aplicam os mesmos métodos neste cenário. No *learning automata* utilizou-se $\alpha = 0.7$, $\omega = 0.1$, 100 passos de tempo e 150 episódios com base no trabalho de Ramos *et al.* (2014). Já para o Q-*learning* com estados utilizou-se $\alpha = 0.8$, $\gamma = 0.9$ $\epsilon = 1.0$ com uma taxa de decaimento de 0.925 por episódio, 100 passos de tempo e 150 episódios, conforme apresentado em Grunitzki *et al.* (2014). Tanto o *learning automata* quanto o Q-learning sem estados utilizaram $|K| = 8$.

4.3. Análise do tempo médio de viagem

Para demonstrar a eficiência dos diferentes métodos, avaliamos o tempo médio de viagem obtido em cada uma das abordagens no problema anteriormente apresentado. Os resultados são apresentados na Tabela 2. A última coluna representa o tempo médio calculado sobre as 1700 viagens. Os resultados dos métodos de AR, por se tratar de métodos não determinísticos, são valores médios de 30 execuções. É possível perceber que os métodos de AR obtiveram os melhores resultados.

Estes resultados são justificáveis pelo fato de que nos métodos de AR, cada veículo aprende de maneira independente qual é a rota que minimiza seu custo de viagem, enquanto que, nos demais métodos, os veículos apenas recebem uma rota pré-computada por uma entidade central. Os métodos de AT desconsideram fatores como autonomia e tomada de decisão individual. Como demonstram os resultados, a distribuição da responsabilidade da escolha de rota pode trazer benefícios para o sistema como um todo. Isto demonstra que nos métodos de AR, a demanda é melhor distribuida sobre a rede viária, de modo que as vias sejam menos sobrecarregadas.

Tabela 2: Tempo médio de viagem em minutos e desvio padrão (entre parênteses) obtidos pelos métodos disponíveis na ferramenta, para a rede OW.

Método	AL	AM	BL	BM	Geral
All-or-nothing	138.00	97.00	128.00	87.00	114.59
Sucessive Averages	101.20	86.36	97.72	82.88	92.78
Incremental Assignment	96.20	83.80	93.40	79.80	88.93
Learning Automata	91.29 (0.65)	72.32 (0.40)	94.76 (1.20)	68.70 (0.49)	82.12 (0.49)
Q-learning sem estados	84.69 (0.45)	68.09 (0.60)	80.03 (0.76)	62.17 (0.50)	74.67 (0.25)
Q-learning com estados	81.25 (0.23)	67.86 (0.35)	78.86 (0.25)	64.32 (0.40)	73.64 (0.23)

Embora os métodos de AT tenham apresentado resultados inferiores aos demais, seus resultados são importantes para avaliar a qualidade de abordagens alternativas. No caso do Q-*learning* com estados, seus resultados, superiores à variante sem estados, podem ser justificados pelo fato de que enquanto nos demais métodos de aprendizado por reforço os motoristas aprendem a partir de um conjunto k de rotas pré-definidas, no Q-*learning* com estados, os motoristas têm a sua disposição uma quantidade infinita de rotas (considerando que eles podem andar em círculos). Esta característica torna a tarefa ainda mais difícil, mas o método se mostra robusto o bastante para encontrar boas soluções nestes domínios.

5. CONCLUSÕES

Este trabalho apresentou uma ferramenta em Java para realizar a tarefa de alocação de tráfego (AT) e aprendizagem de rotas (AR) em redes viárias. O objetivo desta ferramenta é permitir realizar AT e AR a partir da correta definição das entradas do cenário (rede viária, demanda e função de custo). A ferramenta apresenta seis métodos implementados: *all-or-nothing assignment*, *incremental assignment*, *successive averages*, *Q-learning* sem estados, *Q-learning* com estados e *learning automata*. A ferramenta torna a utilização e execução destes métodos transparente, bastando que seus parâmetros sejam devidamente definidos.

Em função da ferramenta ser desenvolvida em Java, é possível utilizá-la nas mais diversas plataformas (Linux, Windows, Mac e etc.), desde que uma máquina virtual Java compatível esteja instalada. Além disso, a paralelização de suas principais rotinas permite que novos métodos sejam desenvolvidos sem que o desempenho da ferramenta seja comprometido. Fato que torna a ferramenta escalável para cenários maiores e mais complexos, além de permitir um melhor aproveitamento do hardware utilizado.

Resultados experimentais em um cenário abstrato demonstram que, entre os métodos inclusos, os de AR apresentaram resultados superiores quando comparados aos métodos clássicos de AT. Trabalhos futuros preveem o desenvolvimento de uma interface visual que permita aos usuários leigos utilizar a ferramenta sem conhecimentos específicos de programação.

AGRADECIMENTOS

Os autores gostariam de agradecer aos revisores por suas sugestões e comentários. Ricardo Grunitzki e Ana L. C. Bazzan são parcialmente apoiados pelo CNPq e FAPERGS. Todos os autores são apoiados pelo ITL/SENAT.

REFERÊNCIAS BIBLIOGRÁFICAS

Arfaoui, A. (1999) Estimation des matrices origine-destination a partir des comptages: etude de quelques modeles. Rep. tec., Laboratoire de Statistiques Théoriques et Appliquées, Université Pierre et Marie Curie.

Buşoniu, L.; Babuska, R. e De Schutter, B. (2008) A comprehensive survey of multiagent reinforcement learning. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 38(2), 156–172.

Grunitzki, R.; Ramos, G. d. O. e Bazzan, A. L. C. (2014) Individual Versus Difference Rewards on Reinforcement Learning for Route Choice. Em: *Proceedings of the 2014 Brazilian Conference on Intelligent Systems (BRACIS 14)*. URL www.inf.ufrgs.br/maslab/pergamus/pubs/grunitzki+2014-bracis.pdf. Forthcoming.

Narendra, K. S. e Thathachar, M. A. L. (1989) *Learning Automata: An Introduction*. Prentice-Hall, Upper Saddle River, NJ, USA.

Ortúzar, J. e Willumsen, L. G. (2001) *Modelling Transport*. John Wiley & Sons, terceira ed.

Ramos, G. de. O.; Grunitzki, R. e Bazzan, A. L. C. (2014) On improving route choice through learning automata. Em: *Proceedings of the Fifth International Workshop on Collaborative Agents – Research & Development (CARE 2014)*, p. 1–12. URL http://www.inf.ufrgs.br/maslab/pergamus/pubs/ramosetal2014care.pdf.

Tumer, K. e Agogino, A. (2006) Agent reward shaping for alleviating traffic congestion. Em: *Workshop on Agents in Traffic and Transportation*. Hakodate, Japan.

Watkins, C. J. C. H. e Dayan, P. (1992) Q-learning. *Machine Learning*, vol. 8(3), 279–292.

Ricardo Grunitzki (rgrunitzki@inf.ufrgs.br)
Gabriel de Oliveira Ramos (goramos@inf.ufrgs.br)
Ana L. C. Bazzan (bazzan@inf.ufrgs.br)
Instituto de Informática, Universidade Federal do Rio Grande do Sul
Av. Bento Gonçalves, 9500 - Jardim Carvalho, Agronomia, Porto Alegre - RS