Davi Guggisberg Bicudo

Aplicação do Simulador de Tráfego MATSim à Cidade de Joinville/SC

Joinville - Brasil

2015

Davi Guggisberg Bicudo

Aplicação do Simulador de Tráfego MATSim à Cidade de Joinville/SC

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do título de bacharel em Engenharia de Transportes e Logística no Centro de Engenharias da Mobilidade da Universidade Federal de Santa Catarina, Campus de Joinville.

Universidade Federal de Santa Catarina – UFSC

Centro de Engenharias da Mobilidade

Bacharelado em Engenharia de Transportes e Logística

Orientador: Prof. Dr. Gian Ricardo Berkenbrock

Joinville - Brasil

2015

Davi Guggisberg Bicudo
Aplicação do Simulador de Tráfego MATSim à Cidade de Joinville/SC/ Davi Guggisberg Bicudo. – Joinville - Brasil, 2015-
127 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Gian Ricardo Berkenbrock

Trabalho de Conclusão de Curso – Universidade Federal de Santa Catarina – UFSC
Centro de Engenharias da Mobilidade
Bacharelado em Engenharia de Transportes e Logística, 2015.

1. Modelagem de transportes. 2. Simulação multi-agentes. 3. MATSim. 4. Joinville. I. Orientador: Prof. Dr. Gian Ricardo Berkenbrock. II. Universidade Federal de Santa Catarina. III. Centro de Engenharias da Mobilidade. IV. Bacharelado em Engenharia de Transportes e Logística V. Título: Aplicação do Simulador de Tráfego MATSim à Cidade de Joinville/SC

CDU 02:141:005.7

Davi Guggisberg Bicudo

Aplicação do Simulador de Tráfego MATSim à Cidade de Joinville/SC

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do título de bacharel em Engenharia de Transportes e Logística no Centro de Engenharias da Mobilidade da Universidade Federal de Santa Catarina, Campus de Joinville.

Trabalho aprovado. Joinville - Brasil, 10 de julho de 2015:

Prof. Dr. Gian Ricardo Berkenbrock
Orientador

Prof.ª Dr.ª Renata Cavion
Membro 1

Prof.ª Dr.ª Simone Becker Lopes
Membro 2

Prof. Dr. Rafael Machado Casali
Membro 3

Joinville - Brasil
2015

Tesouro na terra,
pilares da minha vida.
À família, que somente é,
a maior de todas as dádivas.

Agradecimentos

Os meus agradecimentos mais sinceros a todos que participaram nessa caminhada durante a faculdade e aos que me auxiliaram chegar a ela.

A Deus, pela proteção e amor infinitos.

À minha mãe, que me fez quem eu sou.

À minha família, irmãos, tios, primos, avós, pois tornam a vida mais leve e com significado. Especialmente ao tio Lelo e à tia Mônica, que foram verdadeiras fortalezas sobre as quais pude me apoiar para tudo.

À minha noiva, Josilaine. Anjo, por acreditar em mim sempre e com o teu amor, você me transformou no que eu sou hoje, alguém muito melhor, mais feliz e realizado.

Agradeço ao meu orientador Prof. Gian Berkenbrock por acreditar em mim quando nem eu acreditava e por exigir de mim sempre muito mais do que eu achava ser possível, foi o que me fez ir muito mais longe e concluir esse trabalho.

Aos professores da faculdade, especialmente a todos os professores de Eng. de Transportes e Logística e a todos aqueles que marcaram minha formação, enquanto ser humano (especialmente professor Fabiano) e enquanto profissional.

A todos os servidores da UFSC que fazem este Campus acontecer. Especialmente à minha amiga Amarilis Laurenti.

A todos os verdadeiros amigos formados na faculdade, mesmo aqueles com os quais não tenho mais contato. Sem vocês esse período não teria sido o mesmo. Patrício, Anderson, Willian, Rodrigo, Marcos, Rafa, Duda, Pedro, Michels. Aos amigos e colegas do curso de Eng. de Transportes e Logística pelo companheirismo sem igual.

A todos os amigos e colegas de todos os grupos de extensão nos quais participei, trabalhamos muito juntos.

A todo o IPPUJ, que me recebeu como estagiário voluntário e me forneceu a maioria dos dados necessários a este trabalho. Principalmente ao meu chefe Osmar, que fez com que eu realmente aprendesse durante o estágio e à Viviane.

A todos os funcionários de órgãos públicos que me disponibilizaram ou auxiliaram na coleta de dados, principalmente ao Serede no Detrans.

Ao meu cunhado Marcos por se disponibilizar a digitar 2000 formulários para me ajudar.

À toda a equipe do MATSim, que desenvolve um *software* de excelente qualidade e

livre, por também me auxiliar com importantes dúvidas.

À Senozon AG, empresa Suíça que forneceu uma licença do *software* Via, o que permitiu a análise dos resultados de forma muito melhor e mais fácil.

A todos os amantes do conhecimento, que não precisam de nenhum motivo para buscá-lo além do amor ao saber.

"O que herdaste dos teus pais,
Adquire, para que o possua."
(Goethe, Fausto.)

Resumo

No contexto de uma economia emergente como a brasileira, com uma frota de veículos em considerável expansão, a demanda de mobilidade nas cidades é cada vez maior. Demanda que não tem sido atendida nos últimos anos, evidenciado pelo aumento no congestionamento nas grandes cidades do país. Assim, um bom planejamento de transportes se torna necessário, visto que a expansão do tamanho da malha viária é limitada física e economicamente. A criação de modelos de transporte tem o poder de auxiliar nesse planejamento, e com a evolução da capacidade computacional surge a possibilidade do desenvolvimento de modelos baseados em simulação. Dentre esses modelos, uma categoria é a baseado em agentes independentes, na qual o MATSim, ferramenta utilizada neste trabalho, faz parte. Este trabalho visou, portanto, a criação e simulação de um modelo de tráfego da cidade de Joinville/SC. Os dados de entrada recebidos, que permitiram a criação do modelo, foram provenientes do censo demográfico, da pesquisa origem-destino, de informações de uso de solo e da base georreferenciada da rede viária municipal. Os dados utilizados permitiram a representação da malha viária da cidade e de sua população. Essa última correspondeu, em número de agentes, à quantidade total de moradores do município, e as rotinas dos agentes às rotinas das pessoas da cidade. O resultado da simulação foi avaliado por meio de contagens reais de tráfego. Como resultado final, o modelo proveu dados desagregados da movimentação de todos os agentes que, ao serem agregados, permitem a extração de diversos tipos de informação que por sua vez podem ser relevantes ao planejamento de transportes.

Palavras-chaves: modelagem de transportes. Simulação multi-agentes. MATSim. Joinville.

Abstract

Upon the context of an emergent economy like Brazil, with a fleet of automobiles growing considerably, the demand for mobility in cities is ever bigger. Demand that has not been attended in the last years, shown by the increase in congestion in the country's big cities. Thus, a good transport planning is necessary, given that the expansion of the road network is limited physically and economically. The creation of transport models has the power to assist in this planning, and with the evolution of the overall computing capacity emerges the possibility of the development of models based on simulation. Among these models, one category is based on independent agents, on which MATSim, framework used in this work, is part of. This work aimed, therefore, the creation and simulation of a traffic model of the city of Joinville/SC. The entry data received, that allowed the creation of the model, came from the demographic census, the origin-destination survey, information of land use data and the georeferenced base of the municipal road network. The data used allowed the representation of the city's road network and it's population. The latter corresponded, in number of agents, to the total amount of dwellers in the city, and the routines of the agents to the routines of the city's people. The simulation results were evaluated by means of real traffic counts data. As a final result, the model provided disaggregated data of all agents' movements that, being aggregated, allowed the extraction of diverse kinds of information which in turn might be relevant to transport planning.

Key-words: transport modelling. Multi-agent simulation. MATSim. Joinville.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ANTP	Associação Nacional dos Transportes Públicos
CTA	Controle de Tráfego em Área
Detrans	Departamento de Trânsito de Joinville
DETRAN/SC	Departamento Estadual de Trânsito de Santa Catarina
ESRI	*Environmental Systems Research Institute*
IBGE	Fundação Instituto Brasileiro de Geografia e Estatística
IES	Instituição de Ensino Superior
IPC	Instituto de Pesquisa Catarinense
IPPUJ	Fundação Instituto de Pesquisa e Planejamento para o Desenvolvimento Sustentável de Joinville
IPTU	Imposto Predial e Territorial Urbano
MAS	*Multi-agents Simulation*
MATSim	*Multi-Agent Transportation Simulator*
OD	Origem-Destino
PlanMob	Plano de Mobilidade de Joinville
PLADSTU	Plano Diretor do Sistema de Transporte Urbano
PMJ	Prefeitura Municipal de Joinville
QGIS	*Quantum Geographic Information Systems*
XML	*eXtensible Markup Language*

Sumário

1 Introdução

Diante de um volume de tráfego crescente nas cidades, decorrente do aumento da frota e do incremento das rotinas de atividades dos indivíduos, as sociedades se vêem perante uma demanda cada vez maior por mobilidade nas áreas urbanas. No período entre 2003 e 2012, de acordo com a ANTP (2014), a quantidade de automóveis nas cidades com mais de 60 mil habitantes cresceu 70% e o número de viagens por habitante 27%, enquanto a população apenas 16%. Apesar do crescimento, os governos municipais não vêm tendo sucesso em atender à essa crescente demanda, evidenciado por um aumento de 14% nos tempos das viagens urbanas, chegando à média de 41,7 minutos no deslocamento casa-trabalho nas regiões metropolitanas brasileiras (PEREIRA; SCHWANEN, 2013).

Nesse contexto, a necessidade de solução desses desafios confere um papel fundamental ao planejamento de transportes, que precisa evoluir no passo que os problemas se tornam mais complexos. De acordo com Ortúzar e Willumsen (2011), o planejamento e a implementação tem o poder de mudar o mundo e a modelagem de transporte pode auxiliar nesse processo se adotada como auxílio efetivo na tomada de decisão.

Ainda de acordo com Ortúzar e Willumsen (2011), um modelo é (tradução do autor): "uma representação simplificada de uma parte do mundo real da qual se têm interesse em certos elementos considerados importantes de um ponto de vista particular". Seguindo esta premissa, este trabalho opta em construir um modelo da cidade de Joinville no qual os elementos considerados importantes são as características de seu tráfego.

A partir de um modelo construído, a simulação de cenários, em especial a microssimulação, é capaz de produzir resultados importantes para análises e previsões de tráfego (BALMER, 2007). Com este fim, muitas ferramentas computacionais tem sido desenvolvidas nos últimos anos para simulação de modelos de tráfego (FARINHA, 2013; SILVA, 2011). Uma das ferramentas neste meio é o MATSim (MATSIM, 2015), que vem sendo aplicado recentemente em várias cidades e cenários (FARINHA, 2013), e apresenta características importantes para o presente estudo como a modelagem baseada em agentes e a capacidade de suportar grandes cenários, dentre outras.

O presente trabalho apoiou-se na motivação de construir um modelo de tráfego baseado em agentes para a cidade de Joinville/SC. Buscou-se compreender as características da construção de um modelo multi-agentes e que tipo de resultados poderiam ser obtidos com a sua simulação. Ao longo do processo, estudou-se as etapas necessárias à sua construção de modo que esse abrangesse toda a cidade, de forma simplificada. Ao fim, verificou-se que tipo de informações poderiam ser extraídas da simulação.

1.1 Objetivos

Diante dos desafios apresentados e da oportunidade que se tem com a simulação para apoio ao planejamento, o objetivo deste trabalho está na criação e simulação de um modelo de tráfego da cidade de Joinville/SC com o uso da ferramenta MATSim. Para atingi-lo, os objetivos específicos são:

a) apresentar a ferramenta MATSim para análise do trânsito de Joinville;

b) verificar a viabilidade da aplicação da ferramenta à Joinville;

c) identificar as limitações nos dados de entrada; e

d) verificar que tipo de informações podem ser extraídas da simulação do modelo.

1.2 Metodologia

Afim de alcançar os objetivos propostos, a metodologia adotada foi composta pelas etapas:

a) estudo da ferramenta MATSim;

b) avaliação, tratamento e adaptação dos dados de entrada;

c) construção de um modelo do tráfego da cidade de Joinville utilizando os padrões do MATSim; e

d) simulação e avaliação do modelo.

1.3 Organização do trabalho

Este trabalho está organizado da seguinte forma: o embasamento teórico no segundo capítulo, a construção do modelo da cidade de Joinville no terceiro capítulo, os resultados da simulação do modelo no quarto capítulo e por fim apresenta-se as considerações finais no último capítulo.

2 Fundamentação

A demanda pelo transporte provém da necessidade humana de realizar atividades em diferentes locais e de transportar bens de um local a outro (RANEY, 2005, p. 5). Apesar da necessidade do homem pelo transporte, este também traz consequências negativas, como os congestionamentos, a poluição atmosférica e sonora, bem como os acidentes de trânsito, conforme postula Raney (2005). De acordo com Ortúzar e Willumsen (2011, p. 6 e 7), o planejamento de transportes deve atender a essa demanda de modo a maximizar o bem-estar social, o que inclui a minimização das consequências negativas dos transportes.

Apesar do planejamento de transportes visar atender à demanda das pessoas, essas buscam atender às próprias necessidades individualmente. Por outro lado, o sistema oferece recursos limitados (espaço, capacidade nas vias, etc.). Esta combinação acarreta competição entre os seus usuários (BALMER, 2007, p. 4). Diante desse cenário, Raney (2005, p. 6) afirma que a tecnologia de planejamento de transportes deve considerar o sistema inteiro, levando em consideração as decisões individuais das pessoas e como elas afetam o sistema como um todo, e como os usuários irão reagir às mudanças no sistema.

Nesse capítulo inicia-se pela descrição do modelo clássico de transportes e, a seguir, apresenta-se a simulação baseada em agentes como alternativa ao modelo clássico. Conceitua-se essa alternativa nos termos do MATSim, bem como faz-se a descrição do seu funcionamento e dos seus dados de entrada. A seguir, faz-se uma análise do cenário do trânsito de Joinville e, por fim, revisa-se alguns trabalhos correlatos.

2.1 O processo de quatro etapas

O modelo clássico de transportes, denominado *The Urban Transportation Model System* (MANHEIM, 1979, p. 119), também conhecido como *processo de quatro etapas*, vêm sendo empregado como base do planejamento de transportes e das pesquisas na área desde os anos 60 (ORTÚZAR; WILLUMSEN, 2011, p. 20), tendo sido utilizado como base de comparação para o desenvolvimento de outras abordagens (BALMER, 2007). O modelo busca equilibrar a oferta (rede viária e modos) e a demanda de transportes (necessidade de viagens) através de quatro sub-modelos, ilustrados na Figura 1.

Figura 1 – O modelo de quatro etapas.

Fonte: Balmer (2007, p. 5)

O modelo usa como dados de entrada a rede viária urbana, dados de viagens realizadas entre as diferentes regiões da cidade e projeções futuras para essas viagens. De acordo com Ortúzar e Willumsen (2011, p. 20–22), esses dados são utilizados para estimar o total de viagens geradas e atraídas em cada região da cidade. A seguir, define-se a interação entre essas regiões, onde elas são as origens e os destinos das viagens. Na terceira etapa ocorre a escolha de quais modos serão utilizados para esses deslocamentos. Por fim, na quarta etapa, a alocação do tráfego à rede de transportes. A Figura 1 ilustra essas etapas e os resultados intermediários gerados entre elas.

As limitações deste modelo são conhecidas. Alguns autores como Ortúzar e Willumsen (2011) e Manheim (1979) citam o tratamento limitado das escolhas dos viajantes, a falta de rigor científico, o fato de avaliar somente fluxos de transporte, dentre outras limitações. Balmer (2007, p. 5) sugere, além disso, que neste modelo falta (tradução do autor):

> *Desagregação por viajantes individuais.* O modelo usa fluxos de tráfego, sem discernir o que indivíduos estão fazendo. Essa diferenciação permite a planejadores conectar decisões de viajantes a dados demográficos específicos, tornando suas escolhas mais realistas e comportamentais. *Dinâmica temporal.* O modelo de quatro etapas assume que os veículos nos fluxos de tráfego estão em regime permanente, ou seja, independem do tempo. Dessa forma ignora efeitos que dependem do tempo como efeitos de dispersão de tráfego em horário de pico, congestionamento bloqueando semáforos em verde ou emissões de veículos.

A partir dessas considerações, alguns autores vêm buscando alternativas ao modelo clássico, como a abordagem por agentes.

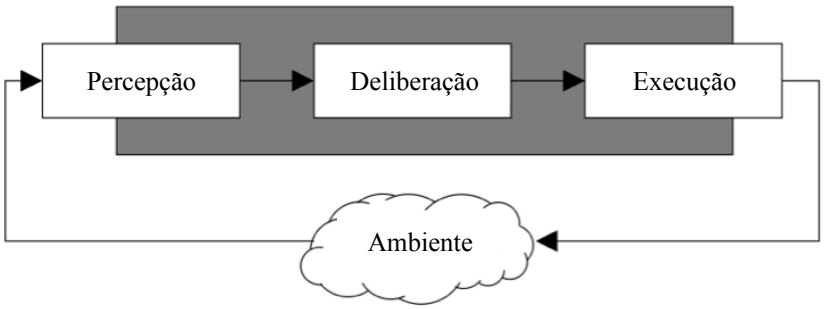
2.2 A simulação de transportes baseada em agentes

De acordo com Raney (2005, p. 1, tradução do autor): "Simulação é uma tecnologia comumente usada na qual um modelo da realidade – como um sistema de transportes e sua dinâmica – é implementado em um computador e executado ao longo do tempo". O autor ainda afirma que a abordagem baseada em agentes pode ser utilizada para a simulação do comportamento individual dos usuários de um sistema de transportes.

Para Weiss (1999, p. 29, tradução do autor), um agente é "um sistema computacional situado em algum ambiente, que é capaz de ação autônoma nesse ambiente de modo a atingir seus objetivos designados". Ainda de acordo com Weiss, um agente tem sensores que o permitem perceber determinados aspectos do ambiente e também um repertório de ações que o tornam capaz de modificar o ambiente. Um agente pode ter, por exemplo, um objetivo único, como maximizar seu desempenho.

Ferber (1999 apud RANEY, 2005) descreve um sistema multi-agentes como "um sistema composto de uma sociedade de agentes em um ambiente compartilhado que age sobre esse ambiente e interage entre si" (tradução do autor). Ferber (1999 apud RANEY, 2005) descreve, também, o conceito de simulação multi-agentes (tradução do autor): "[...] uma *simulação multi-agentes* (MAS na sigla em inglês) é um modelo computacional que representa um conjunto de agentes e seu ambiente, assim como o comportamento, ações e interações dos agentes neste ambiente". Dessa forma, o MAS possibilita o estudo do ciclo de interações da Figura 2.

Figura 2 – Um agente em seu ambiente.

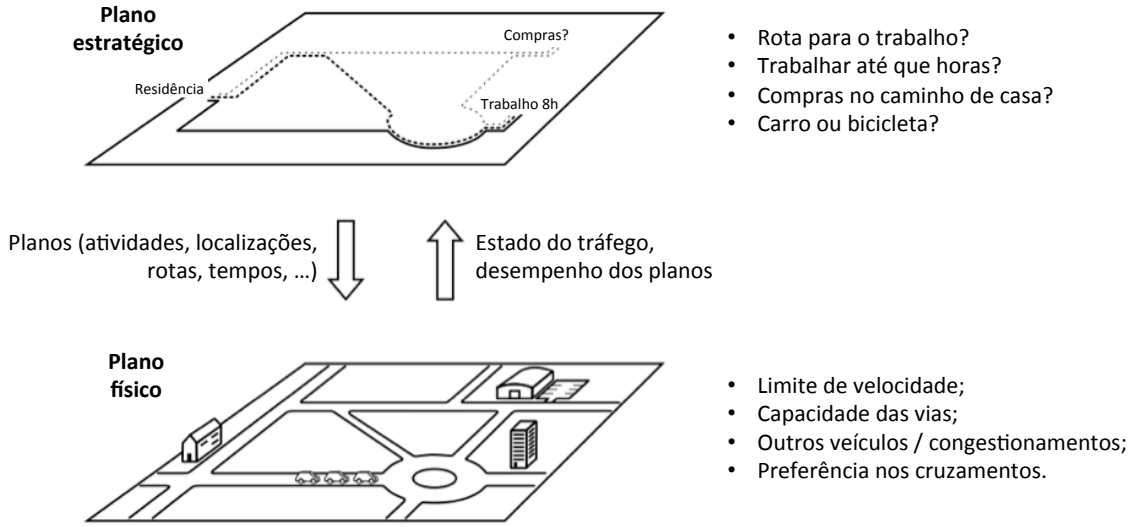


Fonte: Ferber (1999 apud RANEY, 2005, adaptado)

Um sistema baseado em agentes divide-se em duas partes, a dos agentes e a do ambiente. Nesse sistema, cada agente toma decisões de como irá interagir com o ambiente, através da criação de uma estratégia de ação que aplicará sobre o ambiente. No contexto do agente, portanto, existem duas realidades, uma interna que pode ser denominada plano estratégico e outra externa que pode ser denominada plano físico. O plano estratégico é onde o agente planeja sua ação sobre o ambiente externo e o plano físico é o ambiente no

qual o agente irá colocar em prática essas estratégias (RIESER, 2010, p. 20). A Figura 3 ilustra estes dois planos.

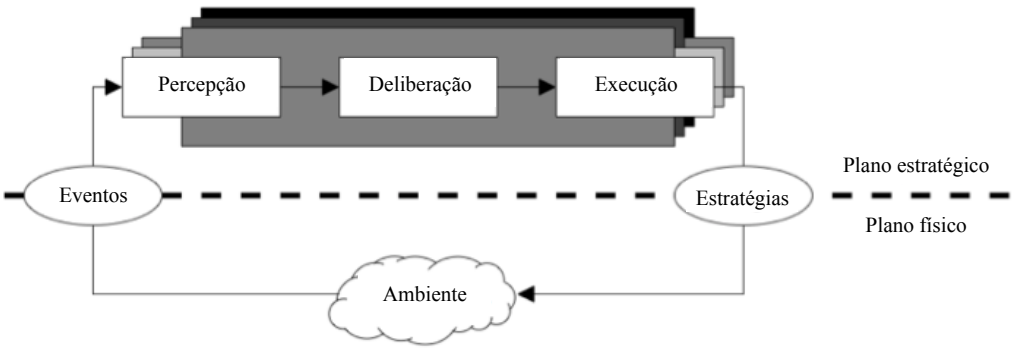
Figura 3 – Os planos mental e físico de um agente.



Fonte: Rieser (2010, adaptado)

Numa simulação multi-agentes, isso ocorre simultaneamente para cada agente. A Figura 4 ilustra o modelo multi-agentes.

Figura 4 – A estrutura do modelo multi-agentes.



Fonte: Raney (2005, adaptado)

Nesse modelo, os agentes planejam sua rotina no plano estratégico de acordo com seus objetivos. Eles tem intenções a cumprir em seu dia-a-dia como: por exemplo, ir trabalhar, ir almoçar, ir ao supermercado, ir à academia e voltar para casa. Para atingir esses objetivos de forma eficiente, esse agente irá se deparar com a necessidade de tomar algumas decisões. Balmer (2007) diferencia as dimensões de decisões que o agente pode

realizar (tradução do autor):

> • Que rota devo seguir para chegar ao trabalho? – *Decisão de rota.*
> • Que modo eu devo escolher para ir ao lago? – *Decisão de modo.*
> • Eu deveria tomar mais uma cerveja antes de ir para casa? – *Decisão de duração de atividade.*
> • Eu deveria ir comprar próximo à minha casa ou no Shopping? – *Decisão de localização.*
> • Quando eu deveria fazer atividade física hoje? – *Decisão de horário de atividade.*
> • Eu deveria ir visitar meu amigo? – *Decisão de tipo de atividade.*
> • Quem eu deveria levar comigo? – *Decisão de composição de grupo.*
> • Devo ir nadar antes ou depois do trabalho? – *Decisão de sequência de atividades.*

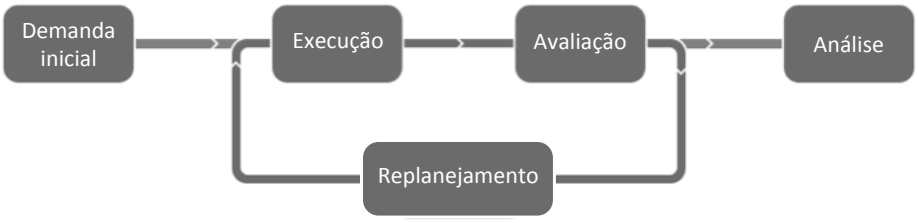
O resultado da combinação dos objetivos do agente com as suas decisões é um *plano* do dia. De modo simplificado, o plano compreende: as atividades que o agente pretende fazer no dia; onde ele pretende realizar essas atividades; e como ele pretende chegar aos locais de realização dessas atividades.

Uma ferramenta que tem sido desenvolvida nos últimos anos e que implementa a abordagem baseada em agentes é o MATSim (2015).

2.3 O MATSim

O MATSim, de acordo com Raney e Nagel (2006), é uma plataforma de simulação de transportes multi-agentes e de larga escala. Seu funcionamento segue o modelo ilustrado na Figura 5.

Figura 5 – Etapas de uma simulação do MATSim.



Fonte: Rieser et al. (2014, adaptado)

Rieser (2010, p. 22) explica o esquema (tradução do autor):

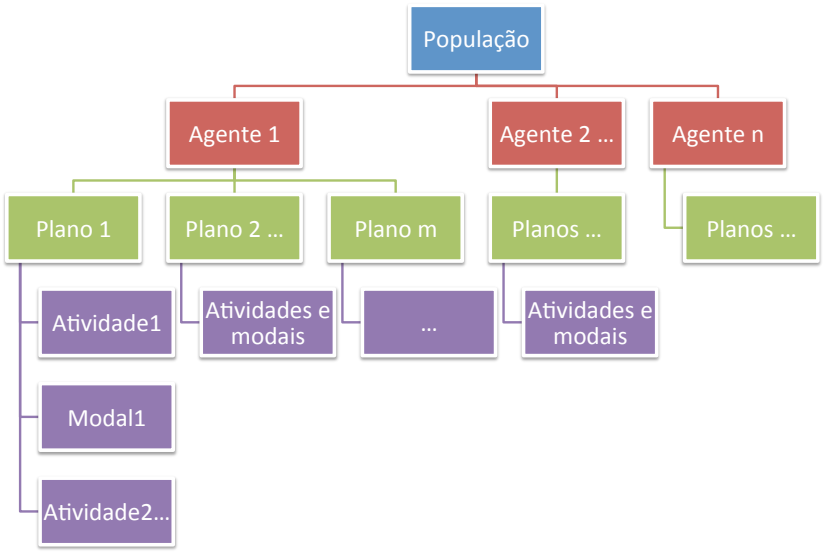
> Ele começa com a *demanda inicial* que descreve um conjunto inicial de planos do dia para a população simulada. Este conjunto inicial de planos

> do dia é então *executado* pelo simulador de tráfego [...]. O desempenho dos planos é avaliado durante a etapa de *avaliação*, tal que os agentes podem tentar otimizar seus planos durante a etapa de *replanejamento*. Os planos são então executados novamente. Este ciclo iterativo continua até que algum critério seja atingido. No final, é possível realizar a *análise* dos dados no que diz respeito ao estado do tráfego ou ao comportamento dos agentes.

2.3.1 Demanda inicial

A demanda inicial corresponde ao comportamento da população, que será simulado (RIESER et al., 2014). Cada agente dessa população deverá ter ao menos um plano, que representa as intenções do agente para o dia. O conjunto dos agentes com seus respectivos planos descreve o comportamento da população. Os dados na população são dispostos de forma hierárquica, onde a população contém uma lista de agentes, um agente contém uma lista de planos e um plano contém uma lista de atividades e modos (RIESER et al., 2014). A Figura 6 ilustra a hierarquia da população.

Figura 6 – Hierarquia de uma população no MATSim.



Fonte: Elaborado pelo autor.

2.3.2 Execução

Os planos do dia de todos os agentes são então simultaneamente executados pelo MATSim no plano físico. O simulador tenta mover os agentes pelo plano físico, o qual limita esses movimentos de acordo com algumas restrições. Rieser (2010, p. 21) enumera algumas dessas limitações (tradução do autor):

• *Limites de velocidade:* um agente não pode trafegar mais rápido que o permitido na via ou o que seu veículo pode alcançar.
• *Capacidade de fluxo da via:* um veículo não pode deixar a via e adentrar um cruzamento juntamente com outros vinte veículos ao mesmo tempo (uma conhecida regra de ouro assume que aproximadamente 2000 veículos podem trafegar por uma via por hora e por faixa).
• *Capacidade de armazenamento da via:* um veículo não pode adentrar uma via se a via já está congestionada por outros veículos.
• *Posição dos outros agentes:* na maior parte dos casos não é desejado simular acidentes de trânsito, portanto veículos não devem colidir com outros veículos.
• *Direito de passagem:* Independentemente de um cruzamento ter semáforo ou não, um agente nem sempre tem o direito de simplesmente seguir adiante.
• *Capacidade do veículo:* se o trem ou ônibus já estiver cheio, o agente terá que aguardar o próximo veículo da linha chegar para poder embarcar.
• *Horário de funcionamento:* um agente não pode fazer suas compras se a loja estiver fechada.

2.3.3 Avaliação

Após a execução, os planos dos agentes são avaliados e recebem uma nota quanto ao seu desempenho. Essa nota é calculada pela equação (BALMER, 2007, p. 31):

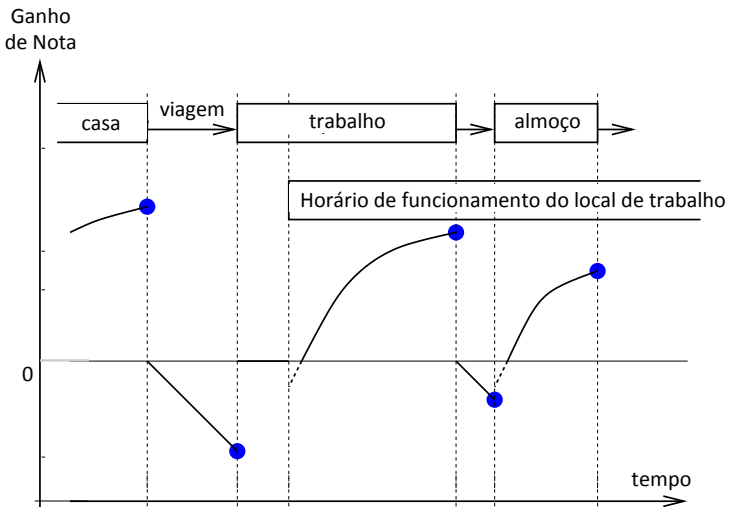
$$V = \sum_i (V_i^{perf} + V_i^{late}) + \sum_i V_j^{leg}$$

onde,

- i é o índice das atividades do plano;
- j é o índice das viagens do plano;
- V_i^{perf} é a nota (positiva) recebida ao executar atividades;
- V_i^{late} é a nota (negativa) recebida por chegar atrasado a atividades; e
- V_j^{leg} é a nota (negativa) recebida ao deslocar-se.

A Figura 7 ilustra a avaliação de um plano, onde o eixo vertical representa o ganho de nota V para cada ação do agente. Nela observa-se ainda a situação especial do ganho nulo no momento em que o agente está esperando o início de uma atividade.

Figura 7 – Exemplo de avaliação de um plano.



Fonte: adaptado de Rieser et al. (2014, p. 26).

2.3.4 Replanejamento

Durante essa etapa, os agentes modificam seus planos com o objetivo de criar novos planos que obtenham melhor nota na execução seguinte. As alterações podem ser de, por exemplo, rotas, durações de atividades ou modos.

Para realizar o replanejamento, o MATSim tem os chamados *módulos de estratégia*, que são compostos de um *seletor de planos* em combinação com zero ou mais *módulos inovativos* (RIESER et al., 2014, em tradução do autor). No início da etapa de replanejamento, o seletor escolhe um dos planos e, caso haja um módulos inovativos disponível, é feita uma cópia do plano escolhido que sofre uma mutação. A seguir o plano é encaminhado à execução.

Os seletores disponíveis são (RIESER et al., 2014, p. 35):

a) *BestScore*: escolhe o plano com a melhor nota;

b) *KeepLastSelected*: seleciona o mesmo plano da última iteração;

c) *SelectRandom*: seleciona aleatoriamente entre os planos disponíveis; e

d) *SelectExpBeta*: escolhe um plano de acordo com a função:

$$p_i = \frac{\exp(\beta_{brain} * S_i)}{\sum_j \exp(\beta_{brain} * S_j)}$$

onde, p_i são os planos, S_i suas notas e o parâmetro β_{brain} é determinado pelo usuário.

Os módulos inovativos são (RIESER et al., 2014, p. 36):

a) *ReRoute*: re-calcula as rotas do plano;

b) *TimeAllocationMutator*: varia aleatoriamente o horário de término das atividades (o limite para a variação é configurável);

c) *ChangeLegMode*: altera o modo utilizado no plano dentre os modos disponíveis;

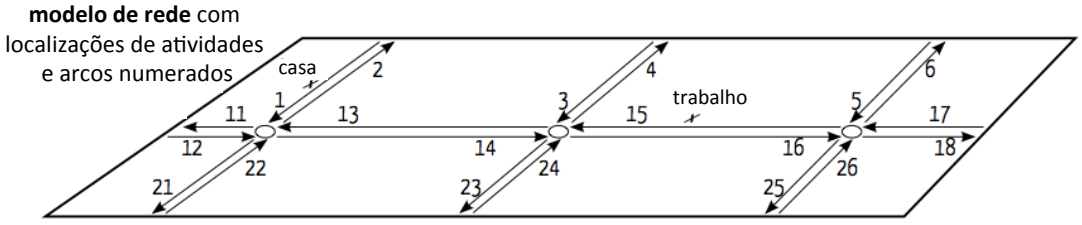
d) *ChangeSingleLegMode*: altera o modo utilizado em apenas um dos deslocamentos; e

e) *SubtourModeChoice*: altera o modo para um circuito no plano, como ir e voltar do trabalho para o almoço.

2.3.5 Análise

O MATSim oferece seus resultados no formato de *eventos*, que identifica tudo que um agente faz ao longo de sua rotina. Os eventos podem ser sair de casa, entrar e sair de um arco, iniciar ou terminar uma atividade, etc. Eles contém todos os dados da etapa de execução, nenhuma atividade de agregação é feita (RANEY, 2005, p. 27). De acordo com Balmer (2007, tradução do autor) "a etapa de execução retorna informações precisas sobre onde um agente se encontra (para cada segundo) e o que está fazendo (por exemplo dirigindo, chegando a um local, entrando num segmento de via e etc.)". A Figura 8 ilustra a correspondência entre o plano executado de um agente e os eventos gerados durante sua execução.

Figura 8 – Eventos gerados durante a execução de um plano.



plano do agente 2:

- permanecer em casa no arco 1 até 7h05
- dirigir pelos arcos 14, 16 e 15
- iniciar o trabalho no arco 15

tipo de evento	horário do evento	id do agente	id do arco	Informação adicional
ActivityEndEvent	07:05:00	2	1	casa
AgentDepartureEvent	07:05:00	2	1	
AgentWait2LinkEvent	07:05:01	2	1	
LinkLeaveEvent	07:05:02	2	1	
LinkLeaveEvent	07:05:02	2	14	
LinkLeaveEvent	07:06:24	2	14	
LinkLeaveEvent	07:06:24	2	16	
LinkLeaveEvent	07:08:51	2	16	
LinkLeaveEvent	07:08:51	2	15	
AgentArrivalEvent	07:10:43	2	15	
ActivityStartEvent	07:10:43	2	15	trabalho

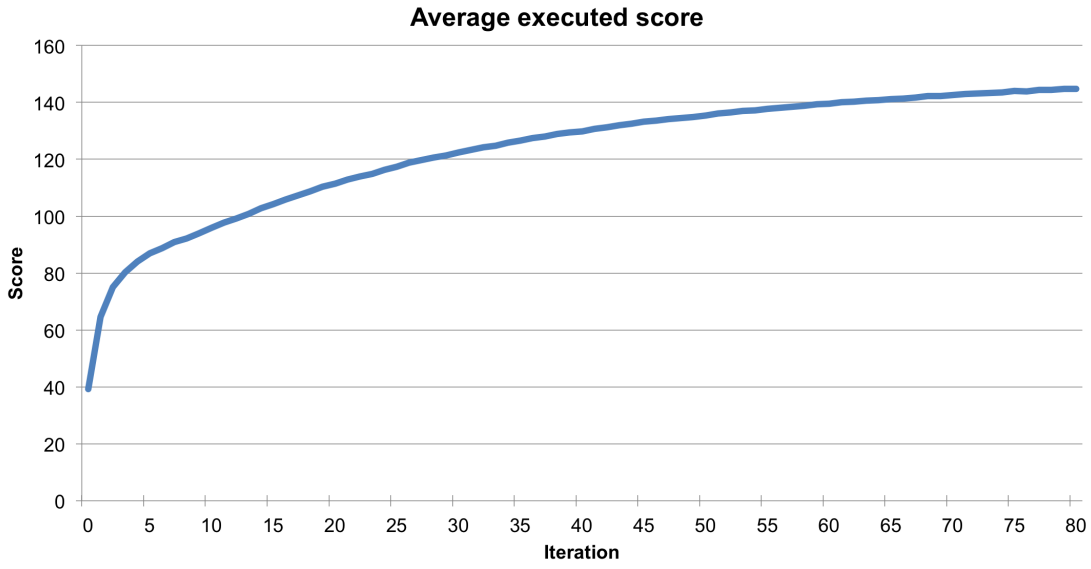
Fonte: adaptado de Rieser (2010, p. 35).

Assim, o resultado da simulação é genérico e desagregado e, para extrair informações de interesse é necessário que se faça sua agregação. O MATSim realiza algumas rotinas de agregação e o usuário também pode fazê-lo por conta própria (RIESER, 2010).

2.3.6 O processo de otimização do MATSim

Durante o processo iterativo do MATSim (Figura 5), conforme descrito nas seções anteriores, os agentes buscam melhorar seus planos criando e executando novas alternativas. De acordo com Rieser et al. (2014, tradução do autor), "o conceito principal do processo de otimização segue os princípios dos denominados *algoritmos (co-)evolucionários*". Algoritmos evolucionários baseiam-se na *teoria da evolução* biológica de Darwin e resolvem problemas de otimização através da melhoria iterativa de "populações" de soluções experimentais (HILLIER; LIEBERMAN, 2006). Essa melhoria ocorre através de mecanismos inspirados na biologia evolutiva como mutação, recombinação e seleção natural (ENGELBRECHT, 2002 apud FOURIE, 2009). No caso do MATSim, a população otimizada é o conjunto de planos de cada agente e não a população de agentes da Figura 6. O algoritmo é chamado co-evolucionário, pois cada agente opera um algoritmo evolucionário independente. O objetivo do MATSim é melhorar, a cada iteração, a média das notas dos planos. A Figura 9 ilustra a evolução típica dessa média em simulações do MATSim.

Figura 9 – Típica curva de evolução da média das notas dos planos em uma simulação do MATSim.



Fonte: adaptado de Rieser et al. (2014, p. 26).

2.3.7 Os dados de entrada

Uma simulação do MATSim necessita de um ambiente (rede de transportes) e de uma população de agentes com seus respectivos planos. O modelo no simulador também pode ser enriquecido com algumas informações adicionais como dados de *uso de solo* e de *contagens de tráfego*. Os dados de uso de solo são úteis para melhor determinar a localização das atividades dos agentes enquanto os dados de contagens servem para avaliação dos resultados da simulação (RIESER et al., 2014, p. 16).

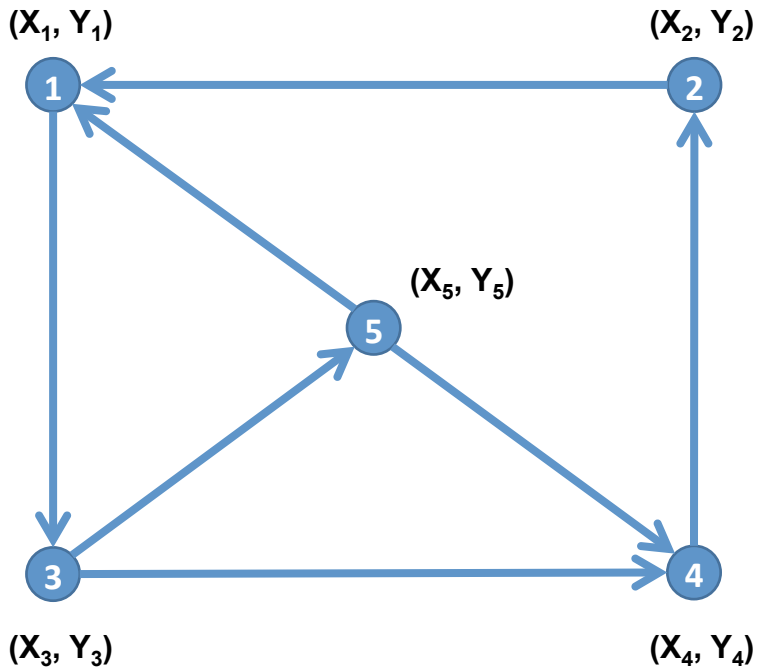
Essa seção irá descrever esses dados de entrada, quanto às suas características, seu formato e seus parâmetros mínimos. Um aspecto geral é de que todos os dados de entrada usados no MATSim devem ser fornecidos no formato *eXtensible Markup Language* (XML) (W3C, 2006). XML pode ser definida como uma metalinguagem utilizada para descrição de outras linguagens (FLYNN, 2006).

2.3.7.1 A rede de transportes

A rede viária é a infraestrutura capaz de atender às viagens geradas pela população. Segundo Hoel, Garber e Sadek (2011), a infraestrutura é a parte fixa de um sistema de transporte e é por onde os usuários entram e saem do sistema. De acordo com a *American Association of State Highway and Transportation Officials* (AASHTO, 2004 apud HOEL; GARBER; SADEK, 2011), uma rede viária é classificada em vias locais, coletoras e arteriais.

No MATSim, a rede de transportes é representada por um conjunto de nós e arcos, onde os arcos representam as vias e os nós as intersecções entre elas. De acordo com Balmer (2007, p. 60), a rede deve ser representada como um *digrafo georreferenciado fortemente conexo*. Um digrafo georreferenciado é um grafo direcionado que tem seus nós atrelados a coordenadas (BALMER; BERNARD; AXHAUSEN, 2005) e será fortemente conexo caso seja possível atingir qualquer nó-destino a partir de qualquer nó-origem (GOLDBARG, 2012). A Figura 10 ilustra um exemplo de um grafo do tipo mencionado.

Figura 10 – Exemplo de digrafo georreferenciado fortemente conexo.



Fonte: elaborado pelo autor.

Para a confecção da rede de transportes no MATSim, é necessário ainda um conjunto de atributos para cada arco, que permitem a representação das vias. De acordo com as especificações do MATSim (RIESER et al., 2014), os atributos requeridos para cada arco são:

a) o comprimento do arco - maior ou igual à distância euclidiana entre seus nós de origem e destino;

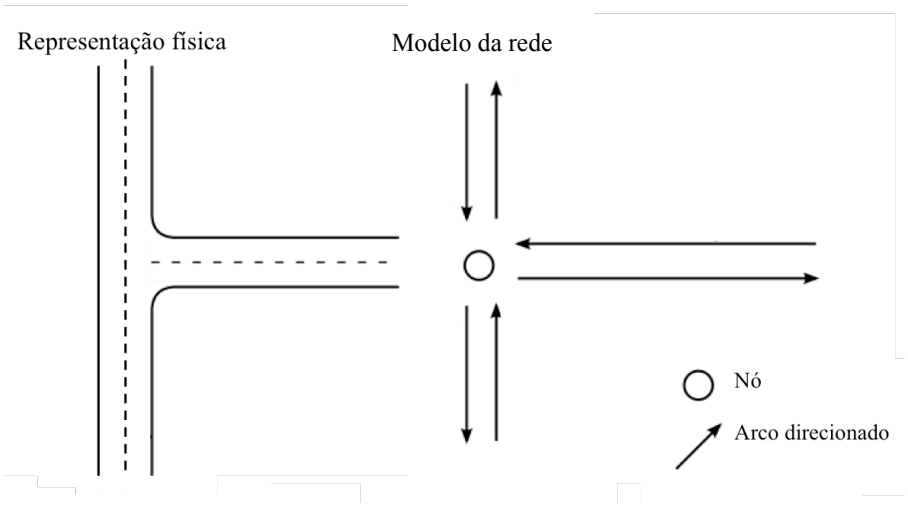
b) o limite de velocidade do arco;

c) o número de faixas de rodagem do arco; e

d) a capacidade máxima de fluxo da via.

A Figura 11 ilustra como uma intersecção viária pode ser modelada. Já a Figura 12 descreve a rede dessa intersecção no formato de entrada do MATSim.

Figura 11 – Exemplo de como uma intersecção real pode ser modelada.



Fonte: elaborado pelo autor.

Figura 12 – Exemplo de definição de uma rede no MATSim.

```
<network name="Exemplo de Rede" xml:lang="pt-br">
  <nodes>
    <node id="1" x="10"    y="0" />
    <node id="2" x="0"     y="0" />
    <node id="3" x="-10"     y="0" />
    <node id="4" x="0"       y="10" />
  </nodes>
  <links capperiod="01:00:00">
    <link id="12" from="1" to="2" length="10" capacity="800"
    freespeed="11.11" permlanes="1" />
    <link id="21" from="2" to="1" length="10" capacity="800"
    freespeed="11.11" permlanes="1" />
    <link id="23" from="2" to="3" length="10" capacity="800"
    freespeed="11.11" permlanes="1" />
    <link id="32" from="3" to="2" length="10" capacity="800"
    freespeed="11.11" permlanes="1" />
    <link id="24" from="2" to="4" length="10" capacity="800"
    freespeed="11.11" permlanes="1" />
    <link id="42" from="4" to="2" length="10" capacity="800"
    freespeed="11.11" permlanes="1" />
  </links>
</network>
```

Fonte: elaborado pelo autor.

Os identificadores (*tags*) da Figura 12 tem os respectivos significados:

a) `<network >`: inicia a descrição da rede;

   – `name`: define o nome da rede;

   – `xml:lang`: define o idioma do arquivo;

b) `<nodes>`: inicia a lista de nós;

c) `<node />`: descreve o nó;

   – `id`: identificador do nó;
   – `x` e `y`: coordenadas do nó;

d) `</nodes>`: finaliza a lista de nós;

e) `<links >`: inicia a lista de arcos;

f) `capperiod`: define a ordem de tempo da capacidade dos arcos, neste caso uma hora;

g) `<link />`: descreve o arco;

   – `id`: identificador do arco;
   – `from` e `to`: nós de origem e destino do arco;
   – `length`: comprimento do arco;
   – `capacity`: capacidade do arco, definida de acordo com o período definido em `capperiod`;
   – `freespeed`: velocidade de fluxo livre do arco;
   – `permlanes`: número de faixas de rodagem do arco;

h) `</links>`: finaliza a lista de arcos; e

i) `<network/>`: finaliza a descrição da rede.

2.3.7.2 A população

A população no modelo do MATSim é a geradora da demanda por viagens, e é descrita como o conjunto de todos os agentes e seus planos de viagem (RIESER et al., 2014).

O comportamento da população (planos dos agentes) compreende a demanda inicial (Figura 5), que é alterada pelo simulador ao longo do seu processo iterativo. Este comportamento é modificado ao longo das iterações do MATSim, durante a etapa de replanejamento. A Figura 13 ilustra um exemplo de arquivo de descrição de população.

Figura 13 – Exemplo de definição de uma população no MATSim.

```
<population>
  <person id="1">
    <plan>
      <act type="home" x="10.0" y="0.0" end_time="08:00:00" />
      <leg mode="car"></leg>
      <act type="work" x="0.0" y="10.0" end_time="17:30:00" />
      <leg mode="car"></leg>
      <act type="home" x="10.0" y="0.0" />
    </plan>
  </person>
  <person id="2">
  ...
  </person>
</population>
```

Fonte: elaborado pelo autor.

Os identificadores (*tags*) da Figura 13 tem os respectivos significados:

a) `<population>`: inicia a descrição da população;

b) `<person />`: descreve um agente;

   – `id`: identificador do agente;

c) `<plan>`: descreve um plano;

d) `< />`: descreve uma atividade;

   – `act type`: tipo da atividade;
   – `x` e `y`: coordenadas da atividade;
   – `end_time`: horário de término da atividade;

e) `<leg />`: descreve um deslocamento;

   – `mode`: identifica o modo do deslocamento;

f) `</leg>`: finaliza a descrição do deslocamento;

g) `</plan>`: finaliza a descrição do plano;

h) `</person>`: finaliza a descrição do agente;

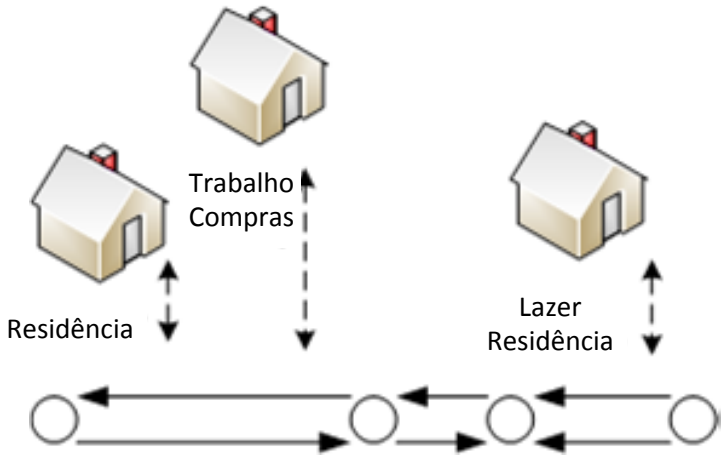
i) `</population>`: finaliza a descrição da população;

Um agente pode ter vários planos, porém apenas um será executado em cada iteração, enquanto os outros planos permanecerão em sua memória e poderão ser executados, alterados ou descartados nas próximas iterações. Os planos dos agentes recebem uma nota a cada iteração, que é atribuída na etapa de avaliação, conforme a seção 2.3.3. A seguir, durante a etapa de replanejamento (seção 2.3.4), outro plano poderá ser escolhido e talvez modificado (RIESER et al., 2014).

2.3.7.3 Uso de solo

As informações de uso de solo não são obrigatórias, mas um complemento que pode ser útil (MATSIM, 2015). As atividades realizadas pelos agentes no MATSim ocorrem nos arcos da rede ou, caso o complemento uso de solo esteja disponível, em um *estabelecimento*. Os estabelecimentos representam construções reais, como lojas, casas e fábricas (MATSIM, 2015). Eles podem conter informações como horário de funcionamento e capacidade máxima de agentes realizando atividades simultaneamente. A Figura 14 ilustra um esquema de estabelecimentos enquanto a Figura 15 ilustra como é o formato dos dados de estabelecimentos no MATSim.

Figura 14 – Esquema de estabelecimentos no MATSim.



Fonte: Adaptado de MATSim (2015)

Figura 15 – Exemplo de definição de um estabelecimento no formato do MATSim.

```
<facility id=”1" x=”10.0" y=”0.0">
  <activity type="home">
    <capacity value="3.0" />
    <opentime day="wkday" start_time="00:00:00" end_time="24:00:00" />
  </activity>
  <activity type="shop">
    <capacity value="10.0" />
    <opentime day="wkday" start_time=”09:00:00" end_time="19:00:00" />
  </activity>
  <activity type="work">
    <capacity value="20.0" />
    <opentime day="wkday" start_time=”08:00:00" end_time=”17:00:00" />
  </activity>
</facility>
```

Fonte: elaborado pelo autor.

Os identificadores (*tags*) da Figura 15 tem os respectivos significados:

a) `<facility >`: inicia a descrição do estabelecimento;

b) `<activity >`: inicia a descrição de uma atividade do estabelecimento;

c) `<capacity />`: descreve a capacidade da atividade;

   – `value`: quantidade máxima de agentes que a capacidade do estabelecimento permite para a atividade;

d) `<opentime />`: descreve o horário de funcionamento da atividade;

   – `day`: identifica os dias de abertura;
   – `start_time`: horário de abertura;
   – `end_time`: horário de fechamento;

e) `</activity>`: finaliza a descrição da atividade;

f) `</facility>`: finaliza a descrição do estabelecimento;

2.4 O trânsito de Joinville

Joinville, uma cidade com cerca de 515 mil habitantes e aproximadamente 364 mil veículos (IBGE, 2010; DETRAN/SC, 2015), enfrenta atualmente desafios na área de mobilidade urbana de elevadas proporções (IPPUJ, 2014).

A origem das diretrizes básicas de mobilidade urbana de Joinville são do ano de 1973 (IPPUJ, 2014) e surgiram a partir da criação do PLADSTU - Plano Diretor do Sistema de Transporte Urbano (JOINVILLE, 1973). Dentre as criações de leis na sequência, destaca-se a criação da Fundação Instituto de Pesquisa e Planejamento Urbano de Joinville (IPPUJ), em 31 de janeiro de 1991. O objetivo de sua criação foi o de colaborar com o poder executivo nos temas ligados ao transporte e uso de solo (IPPUJ, 2014).

Apesar das diretrizes apresentadas pelo PLADSTU, as propostas não foram devidamente acompanhados por medidas executivas nos anos seguintes. Isso está evidenciado pelo fato de que menos de 10% das propostas do Plano Diretor de 73 foram efetivamente executadas (IPPUJ, 2014), potencialmente contribuindo com os graves problemas de mobilidade enfrentados atualmente na cidade.

Com a Lei Municipal nº 261/2008 (JOINVILLE, 2008), que instituiu o Novo Plano Diretor do Município de Joinville, houve o início da criação de um Plano de Mobilidade para a cidade (IPPUJ, 2014).

Uma das fontes de informações utilizadas para a confecção do atual Plano de Mobilidade e Acessibilidade de Joinville (PlanMob) foi a pesquisa origem-destino (OD) de 2010 (IPPUJ; IPC, 2010). De acordo com Hoel, Garber e Sadek (2011, p. 233), uma pesquisa OD permite o entendimento de padrões de viagem. Este tipo de pesquisa colhe

dados sobre: finalidade de viagens; suas origens e destinos; e as modalidades de transporte utilizadas (HOEL; GARBER; SADEK, 2011, p. 233).

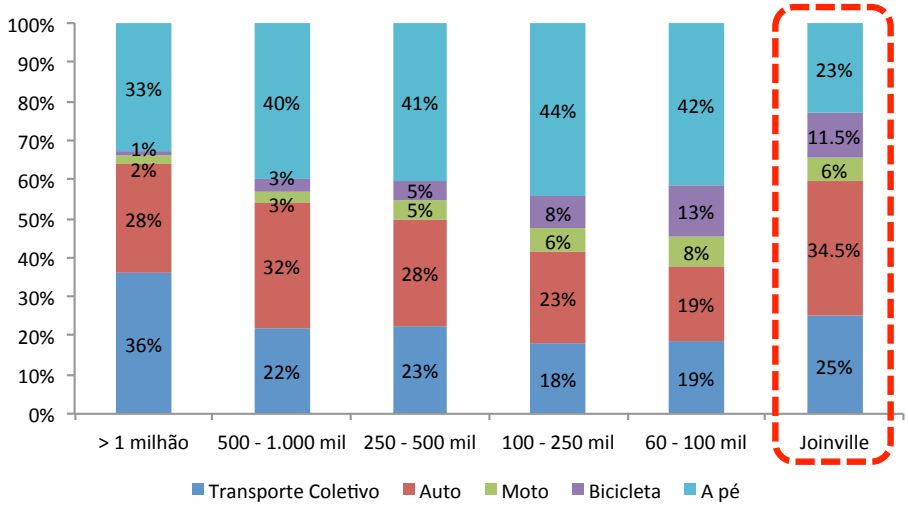
2.4.1 Cenário atual na cidade

As consequências negativas da falta de aplicação da maior parte das propostas do PLADSTU são enumeradas no caderno de diagnóstico do Plano de Mobilidade e Acessibilidade de Joinville (IPPUJ, 2014):

> • Características físicas indiferenciadas, sem hierarquia das vias;
> • Instabilidade do funcionamento da estrutura, causada por período de pico bastante definido de acordo com o horário de funcionamento da indústria, comércio e serviços;
> • Nivelamento, geometria viária, pavimentação, sinalização, estacionamento e deslocamentos significativos de carga pesada no meio urbano devido à localização industrial dispersa;
> • Ligações viárias com descontinuidades físicas notáveis;
> • Impedimento da integração físico/social e do deslocamento viário de caráter mais estrutural, principalmente no que se refere ao transporte coletivo urbano;
> • Necessidade de complementaridade da trama local com melhor articulação interna que incentive o parcelamento, a ocupação mais ordenada e a integração social;
> • Má articulação do sistema com a estrutura, causada principalmente por diretrizes viárias de forma localizada, em áreas já parceladas ou sujeitas ao parcelamento;
> • Vias com desenvolvimento linear com mais de 400 metros ininterruptos;
> • Quarteirões loteados em ângulos menores que 60º com as vias consideradas estruturais;
> • Perímetro urbano extenso, permitindo a urbanização desenfreada e pouco criteriosa.

Além dos problemas acima, algo que também poder ser citado é o fator acima da média nacional de viagens por modo individual motorizado na cidade, como ilustra a Figura 16.

Figura 16 – Divisão modal por densidade demográfica de municípios brasileiros em relação a Joinville.



Fonte: IPPUJ (2014)

2.5 Trabalhos correlatos

Esta seção apresenta alguns trabalhos correlatos de estudos de casos que utilizaram a simulação computacional no planejamento de transportes.

Ittner (2014) aplicou a modelagem para simular, com o *software* AIMSUN (SYSTEMS, 2014) intervenções em uma das principais vias de acesso à cidade de Joinville. Por meio de sua pesquisa pré-experimental, o autor buscou fornecer ao IPPUJ dados de base para decisões e intervenções no trânsito. Também na cidade de Joinville, Silveira (2012) utilizou-se de estratégias para o controle inteligente de semáforos a fim de otimizar o desempenho destes com o propósito de diminuir os efeitos negativos do trânsito, como congestionamentos. Nesse estudo, aplicou a tecnologia de sistemas inteligentes de controle de transporte, bem como controle automatizado de sinais de trânsito e a temporização de semáforos. Para avaliar os resultados, utilizou o *software* ITSUMO (BAZZAN et al., 2010).

Uma das primeiras aplicações do MATSim foram implementadas por Balmer, Axhausen e Nagel (2006). No artigo os autores apresentam os detalhes da modelagem para as cidades de Zurique, Suíça, e Berlim, Alemanha.

Ribeiro (2011) propôs um modelo baseado em agentes com o uso do MATSim para estimativa da geração e da distribuição de viagens interurbanas para a cidade de São Carlos, no estado de São Paulo.

3 O modelo de Joinville

A construção do modelo seguiu as indicações do tutorial do MATSim (2015). Nele, são fornecidas instruções passo-a-passo para que o usuário possa criar um modelo e simular, enquanto compreende as funcionalidades do MATSim.

O tutorial fornece dados da cidade de Zurique, Suíça, para a construção do modelo. Os dados fornecidos são: a rede viária; o censo demográfico; informações de origem-destino; e dados de uso de solo. O usuário é conduzido durante o processo de tratamento e transformação desses dados nos arquivos padrão do MATSim (seção 2.3.7.1). Deste modo, é possível repetir o procedimento para construir um modelo de qualquer cidade, desde que se tenha acesso ao mesmo conjunto de dados. Assim, buscou-se nesse trabalho, obter dados da cidade de Joinville, e seguir os passos sugeridos para confecção do modelo.

Os componentes obrigatórios do modelo são a população, geradora de viagens, no papel de demanda, e a rede viária, receptora de viagens, no papel de oferta. O modelo construído ainda conta com dois componentes complementares, estabelecimentos e contagens de tráfego.

Em resumo, as etapas para confecção do modelo são:

1. confecção da rede viária;
2. confecção do complemento estabelecimentos;
3. confecção da população;
4. confecção do complemento contagens;

Ao longo do processo de construção do modelo de Joinville, muitas etapas de tratamento e adaptação foram executadas, tanto nos dados quanto do próprio modelo base. Este capítulo irá trazer informações sobre os dados de entrada, as características do modelo e a forma como os dados foram processados para confecção do modelo.

3.1 Caracterização dos dados de entrada

Os dados obtidos vieram de diferentes fontes, sendo as principais o IPPUJ, o IBGE e a Prefeitura Municipal de Joinville (PMJ). As subseções a seguir tratam de caracterizar esses dados.

Os dados obtidos e utilizados para a confecção do modelo de Joinville foram:

a) rede viária municipal (IPPUJ, 2015);

b) pesquisa origem-destino do município de Joinville do ano de 2010 (IPPUJ; IPC, 2010);

c) censo demográfico 2010 (IBGE, 2010);

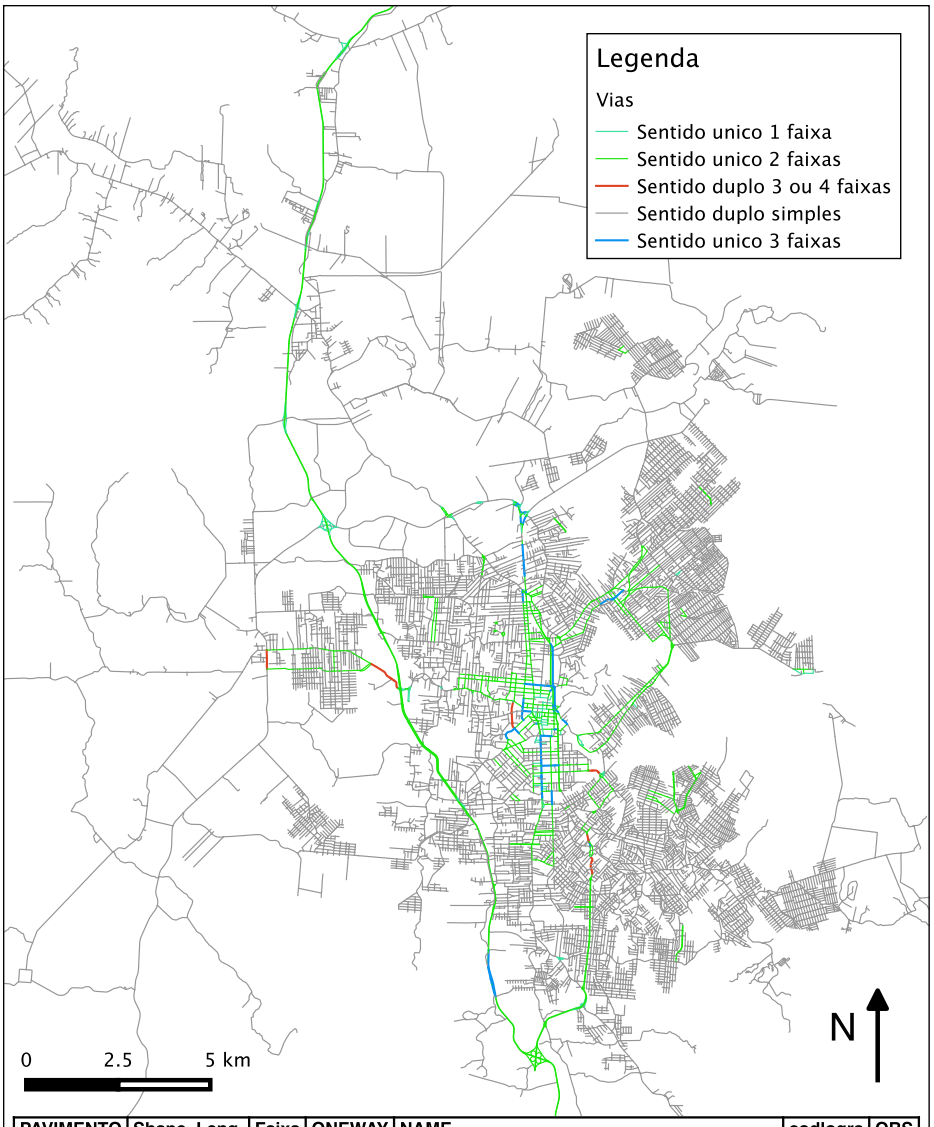
d) seção do banco de dados do Sistema de Gestão Cadastral (SGC) da PMJ (2015); e

e) contagens de tráfego (PMJ, 2015).

3.1.1 A rede viária municipal

Para confecção da rede de transportes, de acordo com as especificações mencionadas na seção 2.3.7.1, é necessário uma fonte de dados que provenha uma representação existente da rede viária de Joinville. A Unidade de Pesquisa e Documentação do IPPUJ forneceu os dados necessários para a confecção da rede, um arquivo no formato *shapefile* (ESRI, 1998) da rede viária da cidade. A Figura 17 ilustra os dados recebidos, nela mostra-se as vias categorizadas por faixas e sentido e, na parte inferior da figura, a tabela com algumas entradas de dados que exemplificam as informações que podem ser extraídas da rede.

Figura 17 – Representação da rede obtida junto ao IPPUJ.



PAVIMENTO	Shape_Leng	Faixa	ONEWAY	NAME	codlogra	OBS
Asfalto	30.6118497468	2	None	HABIB FARAH	4664	None
Asfalto	160.752709148	2	None	HABIB FARAH	4664	None
Asfalto	68.0801747091	3	FT	AVENIDA DOUTOR ALBANO SCHULZ	4910	None
Saibro	109.32774777	2	None	REINOLDO MELZER	9991	None
Asfalto	40.373463225	2	None	PROFESSOR PEDRO VIEIRA	7289	None
Asfalto	92.4540714419	2	None	SD 20050	7289	None

Fonte: Elaborado pelo autor. Dados do IPPUJ (2015).

3.1.2 Pesquisa origem-destino

Os dados da OD foram fornecidos no formato de duas planilhas, contendo em uma os dados socieconômicos das famílias e na outra a característica dos seus deslocamentos. Para este trabalho foi utilizada principalmente a segunda. A Tabela 1 exemplifica o formato da planilha de dados socieconômicos. A Tabela 2 exemplifica os dados de deslocamentos. As

duas tabelas referem-se à mesma família e foram representadas aqui de maneira transposta para melhor visualização.

Tabela 1 – Uma família no banco de dados socieconômico da pesquisa origem-destino.

Área	20	20
Bairro	adhemar garcia	adhemar garcia
Nº do cadastro	116	116
Nº do morador	1	2
Grau de parentesco	Chefe de família	Cônjuge
Idade	18 a 24 anos	25 a 34 anos
Sexo	Masculino	Feminino
Estuda	superior/universitário	superior/universitário
Escolaridade	Superior incompleto	Superior incompleto
Ocupação principal	Assalariado com carteira de trabalho	Assalariado com carteira de trabalho
É deficiente físico?	Não	Não
Renda mensal	De 1 a 2 SM (R$511,00 a R$1.020,00)	De 1 a 2 SM (R$511,00 a R$1.020,00)

Fonte: Fornecido pelo IPPUJ

Tabela 2 – Uma família no banco de dados de deslocamentos da pesquisa origem-destino.

Área	20	20	20	20	20	20
Bairro	adhemar garcia	adhemar garcia	adhemar garcia	adhemar garcia	adhemar garcia	adhemar garcia
Nº do cadastro	116	116	116	116	116	116
Nº do morador	1	1	1	1	2	2
Grau de parentesco	Chefe de família	Chefe de família	Chefe de família	Chefe de família	Cônjuge	Cônjuge
Bairro de saída	adhemar garcia	distrito industrial	adhemar garcia	itaum	adhemar garcia	centro
Bairro de chegada	z.industrial	adhemar garcia	itaum	adhemar garcia	centro	adhemar garcia
Motivo de saída	Residência	Trabalho	Residência	Escola/curso	Residência	Trabalho
Motivo de chegada	Trabalho	Residência	Escola/curso	Residência	Trabalho	Residência
Horário de saída	04_50_00	14_55_00	18_40_00	22_30_00	06_30_00	15_20_00
Horário de chegada	05_10_00	15_15_00	18_50_00	22_40_00	07_15_00	18_00_00
Modos de locomoção	Motocicleta	Motocicleta	Automóvel	Automóvel	Ônibus municipal	Ônibus municipal

Fonte: Fornecido pelo IPPUJ

Ao longo da modelagem, constatou-se algumas limitações dos dados da pesquisa origem-destino de Joinville, as principais são:

a) não há dados para todos os bairros da cidade;

b) algumas entradas não estão no padrão do banco de dados;

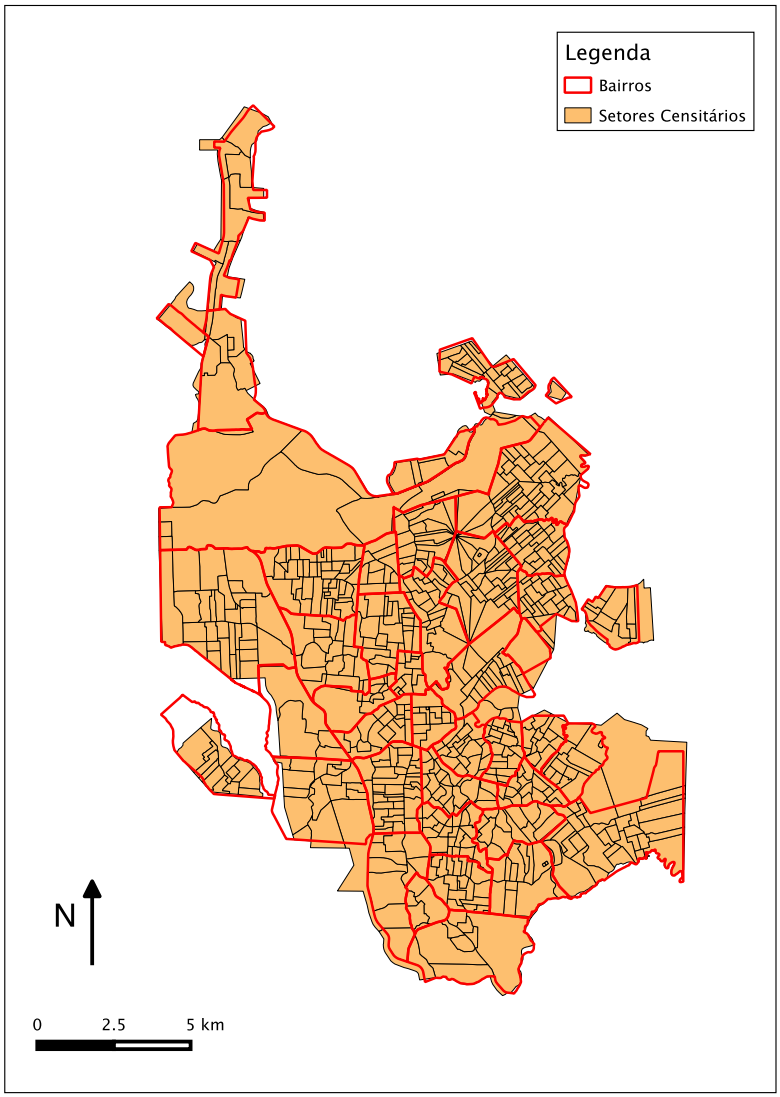
c) alguns deslocamentos são inconsistentes;

A seção 3.4.2 detalha algumas das inconsistências da OD.

3.1.3 Censo demográfico 2010

Os dados do Censo (IBGE, 2010) foram obtidos no formato *shapefile* e estavam segmentados por setor censitário e idade, ou seja, para cada setor censitário da cidade estava disponível a informação do total de habitantes no setor para cada ano de idade. Cada setor censitário corresponde à menor unidade divisível de recenseamento, divididos dentro dos bairros do município de acordo com área e população. A Figura 18 ilustra a disposição dos setores censitários em Joinville.

Figura 18 – Setores censitários de Joinville. As linhas vermelhas demarcam os bairros.



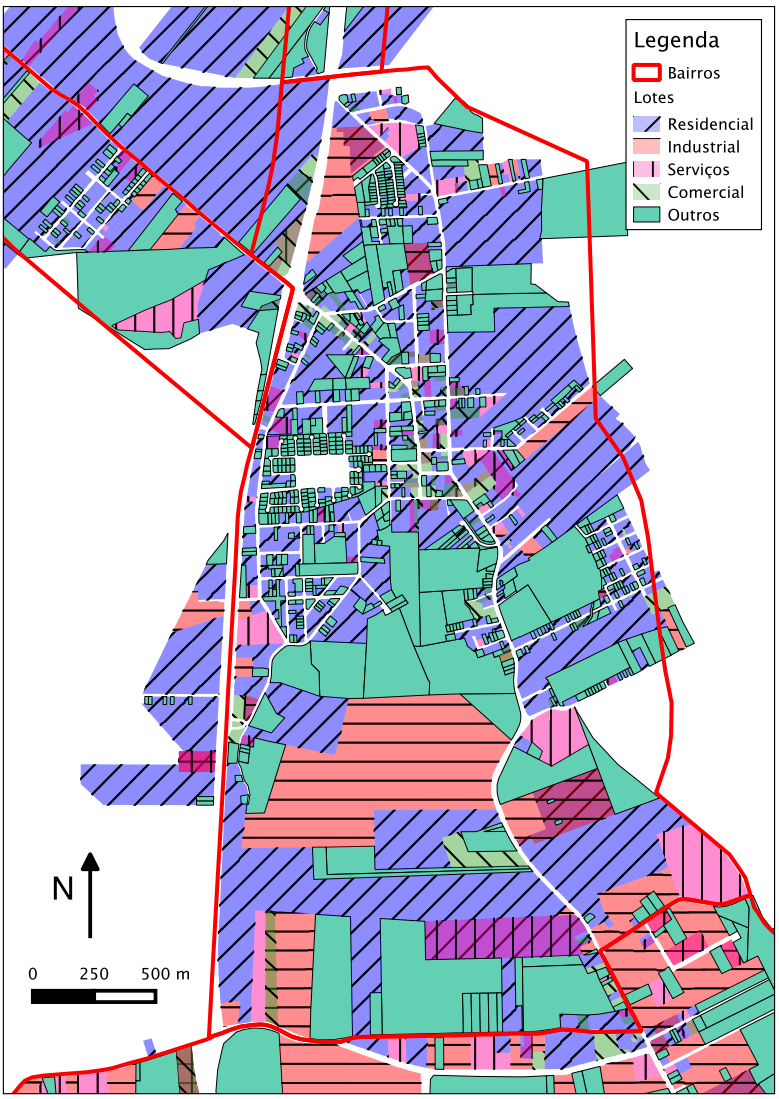
Fonte: elaborado pelo autor. Dados do IBGE (2010) e IPPUJ (2015).

3.1.4 Sistema de Gestão Cadastral da PMJ

O SGC é onde a PMJ mantém o cadastro de todos os lotes da cidade. O sistema inclui dados como área do lote, área construída, localização, tipo de edificação, número de unidades autônomas, tipo das unidades autônomas, registro de alterações no lote, proprietário, valor do Imposto Predial e Territorial Urbano (IPTU), dentre outras.

Apenas uma seção do banco de dados foi disponibilizada, contendo as informações dos tipos de uso dos lotes, além da própria localização e geometria dos mesmos num arquivo *shapefile*. A Figura 19 ilustra, como exemplo, o conjunto de lotes do bairro Pirabeiraba.

Figura 19 – Representação dos lotes do bairro Pirabeiraba de acordo com os principais usos.



Fonte: Elaborado pelo autor. Dados do IPPUJ (2015).

A informação de tipos de uso dos lotes consiste na quantidade de unidades autôno-

mas existentes nos lotes categorizadas por tipo de uso. Os possíveis usos são: residencial; comercial; industrial; serviços; saúde; ensino; cultural; religioso; financeiro; institucional; rural; e baldio.

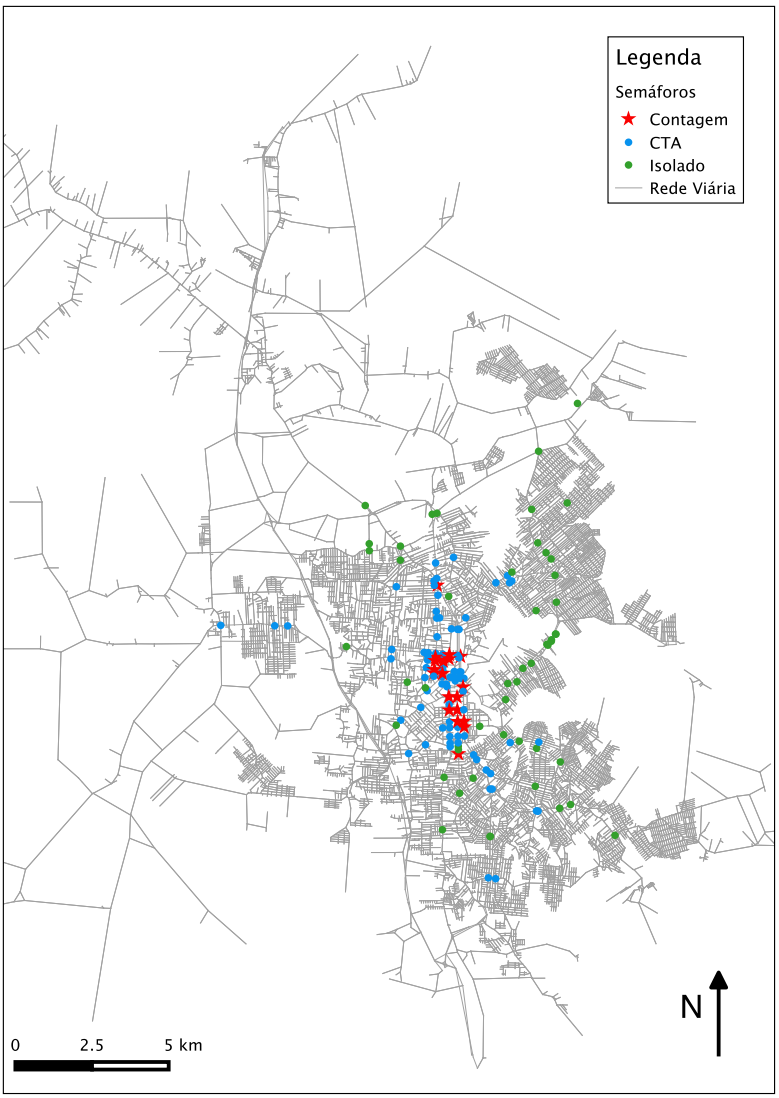
3.1.5 Contagens de tráfego

As contagens de tráfego são importantes na etapa de avaliação da simulação, pois permitem que se avalie quão próximos da realidade estão as vazões de tráfego.

Os dados de contagem foram fornecidos pelo Departamento de Trânsito de Joinville (Detrans). As informações provém do sistema de Controle de Tráfego em Área (CTA), que é um sistema de controle semafórico centralizado instalado em Joinville a partir de 1995 (IPPUJ, 2014). Conforme informado pelo responsável do CTA no Detrans, originalmente todos os semáforos do CTA continham laços indutivos que realizavam as contagens de tráfego automaticamente, porém hoje somente alguns laços ainda estão em funcionamento adequado. Por essa razão, apenas 20 dos 101 semáforos do CTA ainda realiza contagens, menos de 20% do total.

A Figura 20 ilustra os cruzamentos semaforizados de Joinville e classifica-os entre os que fazem parte do CTA e realizam contagens, os que fazem parte mas não estão mais aptos às contagens e os demais semáforos da cidade.

Figura 20 – Mapa com a localização dos semáforos de Joinville e pontos de contagem.



Fonte: Elaborado pelo autor. Dados do IPPUJ (2015).

3.2 A construção do modelo

A partir da compreensão dos dados de entrada, pôde-se construir cada um dos elementos do modelo. As sessões a seguir dão detalhes dos procedimentos realizados para confecção de cada um dos componentes do modelo, suas características e do tratamento dos dados utilizados.

3.3 A rede de tráfego

A rede fornecida não continha todos os atributos necessários ao MATSim (2015), desse modo foram necessários procedimentos para determinação dos atributos faltantes

(seção 2.3.7.1).

O primeiro passo foi completar a informação de sentido das vias. Os dados necessários não existiam em registro no IPPUJ. A sugestão da Unidade de Mobilidade da Fundação, responsável pelas alterações dos sentidos das vias na cidade, foi a verificação visual dos sentidos das vias com o uso da ferramenta Street View (GOOGLE, 2015). A partir dessa verificação, os profissionais do setor fizeram a confirmação dos resultados a partir de sua experiência.

A obtenção do número de faixas foi feita também através de verificação visual com o uso da ferramenta Street View, visto que esses dados também não existiam em registro no IPPUJ. Para realizar essa análise, as vias foram primeiramente divididas em dois grupos. O primeiro grupo consistiu no conjunto das vias de sentido único mais as vias arteriais de sentido duplo. O segundo grupo com as demais vias – coletoras e locais de sentido duplo, as quais correspondem à grande maioria das vias da cidade. Devido ao grande número de vias a verificar, foi necessário realizar uma simplificação onde todas as vias do segundo grupo teriam sempre duas faixas, uma em cada sentido. As demais vias foram verificadas uma a uma.

A seguir determinou-se as velocidades de operação. Para isso, usou-se como base os valores propostos por Dowling (1997), demonstrados na Tabela 3.

Tabela 3 – Velocidades de fluxo livre para vias urbanas de acordo com a classe funcional em km/h.

Tipo de área	**Expressa**	**Arterial**	**Coletora**	**Local**
Região central	72	64	56	50
Urbana	80	72	64	56
Suburbana	88	80	72	64

Fonte: Adaptado de Dowling (1997, p. 84).

Como não se tinha a informação da região das vias, aproximou-se a todas como sendo *urbanas*, exceto rodovias que foram classificadas como *suburbanas*. A informação de classe funcional também não estava disponível, então adotou-se a aproximação de que todas as vias de duas ou mais faixas por sentido seriam classificadas como coletoras e as demais como locais. A exceção foi novamente as rodovias, onde a BR 101 foi classificada como *expressa* e as demais rodovias como *arteriais*.

O último atributo a se determinar foi o fluxo de saturação. Na Tabela 4 verifica-se os valores padrão sugeridos por Dowling (1997).

Tabela 4 – Capacidade de vias urbanas em veic/h de acordo com classe funcional.

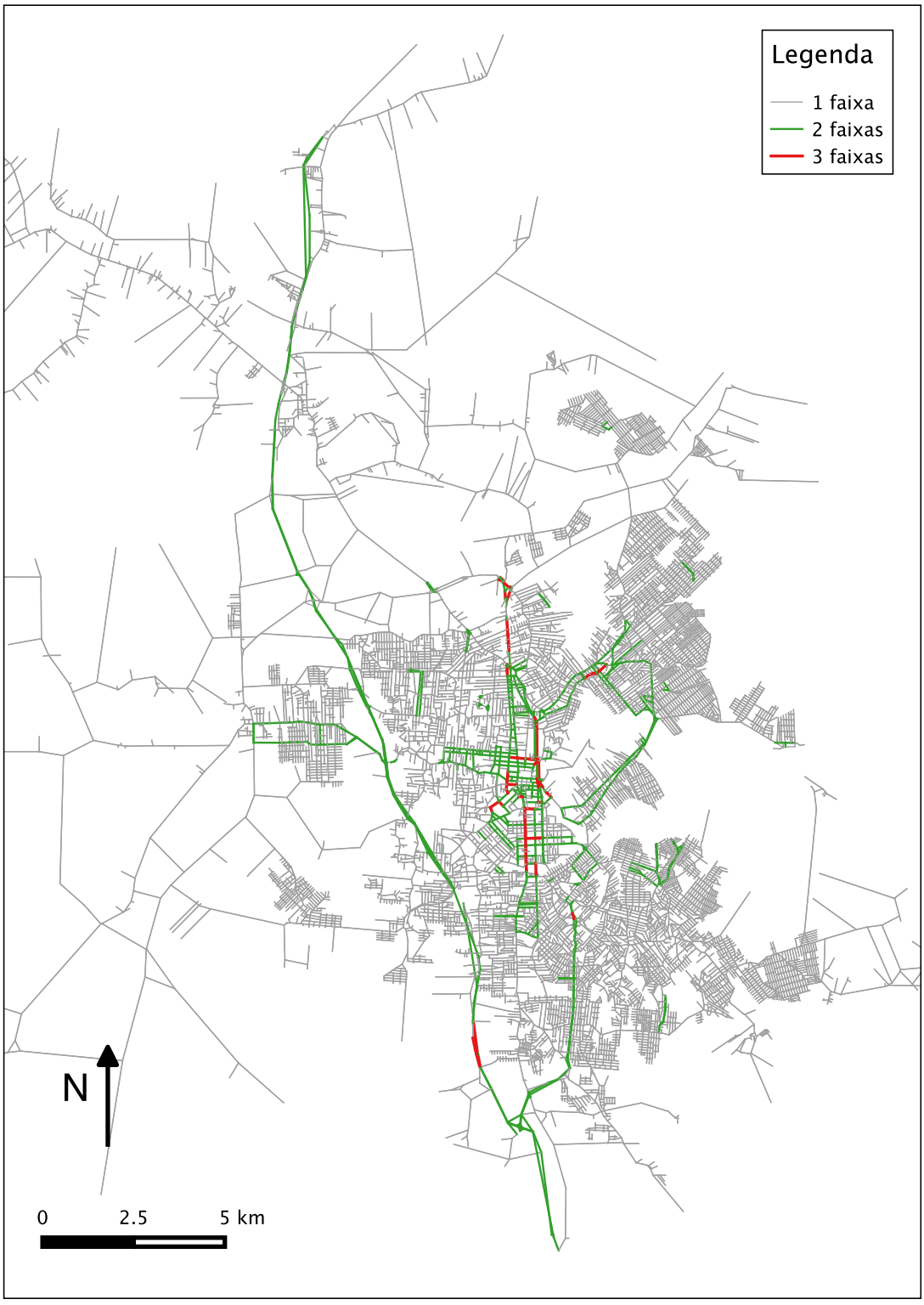
Classe funcional	**Tipo da área**	**Capacidade por faixa**
Via expressa	Urbana	2000
Arterial sem conversão à esquerda	Suburbana	850
	Urbana	750
	Região central	650
Arterial com conversão à esquerda	Suburbana	750
	Urbana	700
	Região central	600
Coletora	Urbana	550

Fonte: Adaptado de Dowling (1997, p. 84).

Uma aproximação para adequar os dados disponíveis às sugestões propostas pela Tabela 4 também foi necessária. Para aproximação dos valores de fluxo de saturação foi utilizado um critério similar ao anterior. Todas as vias de sentido único de duas ou mais faixas foram classificadas como arteriais sem conversão à esquerda, as vias de sentido duplo de duas ou mais faixas por sentido como arteriais com conversão à esquerda e as demais como coletoras. Quanto às rodovias, a BR 101 foi definida como expressa e as demais como suburbanas arteriais com conversão à esquerda visto que são mão-dupla.

Concluída a atribuição dos arcos da rede, prosseguiu-se para a conversão dos dados para o formato do MATSim. Esse processo apresentou dificuldades pois a rede não se tratava de um grafo direcionado ou mesmo conexo (seção 2.3.7.1). Os detalhes da conversão estão no Anexo A. Após a conversão, por fim obteve-se a rede pronta para a simulação. A Figura 21 ilustra o resultado final.

Figura 21 – Rede resultante.



Fonte: elaborado pelo autor.

3.4 A população

A informação de uma população utilizada pelo MATSim é a rotina planejada de cada agente, ou seja, onde e quando pretende realizar cada atividade do seu dia e como pretende se deslocar. Dados socieconômicos podem ser úteis na modelagem da demanda e para análises agregadas mas não são exigidos pelo simulador. Nesse modelo, os agentes serão formados por uma rotina planejada e sua idade, o único dado socieconômico adicionado.

O MATSim simula toda a população, portanto no modelo é criado aproximadamente 510 mil agentes, valor próximo à população total da cidade de Joinville. A diferença de 5 mil em relação à população da cidade (seção 2.4) existe por causa de uma perda de dados durante o tratamento descrito na seção 3.4.1.

Para a construção da população foram feitas algumas considerações a fim de simplificar o modelo e aderir às especificações do MATSim. As considerações foram:

a) o MATSim tem por padrão as atividades moradia, trabalho, lazer, compras e educação (o que não corresponde a todos os tipos de viagem possíveis);

b) no modelo, a atividade educação foi dividida em universidade para agentes acima de 18 anos e educação básica para os demais. Isso para modelar melhor o tráfego próximo às Universidades;

c) os agentes tem uma localização determinada de moradia e trabalho, o restante das atividades em sua rotina é realizada num *estabelecimento* sorteado numa área circular ao redor da última atividade realizada.

d) o MATSim não simula caronas. Portanto, em todos os deslocamento a carro considera-se que a pessoa dirige seu carro sozinha e não leva ninguém consigo. Isso acarreta uma possível distorção na qual agentes com menos de 18 poderiam dirigir, essa possibilidade se deve à OD não diferenciar motoristas de passageiros nos deslocamentos à carro. Uma alternativa seria remover todos os agentes menores de 18 anos mas optou-se por priorizar os deslocamentos em vez dessa possível inconsistência;

e) o MATSim não simula troca modal ao longo do dia, então neste trabalho todas as pessoas que teriam troca modal ficam somente com um modo. Caso ela tenha algum deslocamento a carro, esse será o modo determinado para toda sua rotina;

f) o MATSim simula somente carros, que é o que será considerado nesse trabalho. Todas as pessoas que não se deslocam por carro foram removidas da simulação.

Partindo destas considerações, o processo da geração da população consistiu nas seguintes etapas:

a) criação de um censo sintético da cidade;

b) tratamento e transcrição dos dados de deslocamento da OD;

c) criação de um arquivo com informações de uso de solo;

d) sintetização da população a partir das três informações anteriores;

3.4.1 Censo sintético

Para a construção do censo sintético, seguiu-se um procedimento de agregação e desagregação de dados demográficos. A agregação dos dados consistiu nas etapas:

a) os dados do Censo, que são divididos por idade e setores censitários, foram agregados por bairros e faixas de idade;

b) cálculo da proporção de habitantes por faixa de idade por bairro;

c) os dados de deslocamento da OD permitem saber a contagem de pessoas por bairro que trabalha e, caso trabalhe, em que bairro trabalha. É feito assim uma agregação desses dados de origem-destino de trabalho por bairro;

d) cálculo, por bairro, da proporção de pessoas que trabalham em cada bairro;

e) cálculo da proporção de pessoas que trabalham em cada bairro para toda a cidade;

A partir dos dados agregados, prosseguiu-se para a criação do censo sintético realizando a desagregação segundo as etapas, realizando-as bairro por bairro:

a) decisão da faixa de idade do agente a partir de um número aleatório gerado que correspondesse com a faixa de proporção calculada na agregação;

b) decisão se seria um *trabalhador* ou não a partir de um número aleatório gerado;

c) caso seja trabalhador, efetuar a decisão do bairro de trabalho a partir de um outro número aleatório gerado. A decisão ocorre de acordo com as proporções calculadas anteriormente;

   – no caso das pessoas que vivem em bairros de origem nos quais a pesquisa origem-destino não tem informações, a decisão é feita de acordo com as proporções da cidade.

d) por fim, a pessoa recebe um código de acordo com seu bairro de moradia e bairro de trabalho e é salva no arquivo do censo sintético.

Uma amostra do arquivo resultante é mostrado na Tabela 5.

Tabela 5 – Amostra de dados do censo sintético resultante.

id	**age**	**homeNeighbourhood**	**workNeighbourhood**
1011321	18 to 24	adhemar_garcia	pirabeiraba
1011422	45 to 59	adhemar_garcia	zona_industrial_norte
1010003	45 to 59	adhemar_garcia	-1
1010004	6 to 9	adhemar_garcia	-1

Fonte: Elaborado pelo autor

Os três primeiros dígitos do *id* representam o bairro de moradia, os três dígitos seguintes representam o bairro de trabalho, sendo "000" atribuído para pessoas que não trabalham. Os dígitos seguintes são próprios de cada pessoa. O valor -1 no bairro de trabalho também significa que a pessoa não trabalha.

O procedimento foi executado numa rotina programada na linguagem Python e se encontra no Anexo D.

3.4.2 Tratamento dos dados de deslocamento da OD

Para que o MATSim possa processar uma pessoa da OD, sua rotina deve estar completa, iniciando e terminando na residência e sem inconsistências nos horários. Portanto para enquadrar a pesquisa origem-destino de Joinville no formato proposto pelo tutorial do MATSim, e de acordo com as considerações feitas, foi necessário, além da simples mudança de formato, uma correção dos dados da OD pois várias inconsistências foram identificadas. A análise dos dados permitiu a definição de 11 tipos de inconsistência:

a) *neighbourhoodException* - caso o agente tenha algum bairro de origem ou destino que não se encontra na lista de bairros da cidade. Exemplo: z.industrial (Tabela 2, linha 7, coluna 2);

b) *arrivalTimeFormatException* - caso o formato de algum horário de chegada seja inconsistente. Exemplo: "dia_seguinte";

c) *departureTimeFormatException* - similar ao erro acima porém para horários de saída;

d) *lastActivityArrivalException* - caso o último deslocamento não tenha como destino a atividade "Residência";

e) *firstActivityDepartureException* - caso o primeiro deslocamento não tenha como origem a atividade "Residência";

f) *activityChainException* - caso falte algum deslocamento na rotina do agente. Exemplo: residência-trabalho e depois compras-residência, falta um deslocamento trabalho-compras;

g) *activityTimeSequenceException* - caso algum deslocamento tenha horário de saída anterior ao horário de chegada do deslocamento anterior;

h) *activityTimeOrderException* - caso a sequência dos deslocamentos ordenados por horário de saída seja diferente da sequência dos deslocamentos ordenados por chegada;

i) *activityTimeBoundsException* - caso o horário de saída de algum deslocamento seja posterior ao horário de chegada do mesmo deslocamento;

j) *activityArrivalTimeLimitException* - para retornos à casa antes das 5h da manhã, pois pode significar que a rotina está virando a noite e portanto o deslocamento pertence ao dia seguinte;

k) *activityDepartureTimeLimitException* - para saídas antes das 3h da manhã, para evitar deslocamentos de madrugada pois não se sabe ao certo se pertencem ao dia da rotina ou ao dia seguinte.

A primeira tentativa de transcrever os dados da OD foi sem nenhum tratamento, apenas descartando todas as entradas que apresentavam alguma inconsistência. O resultado foi que, de 8513 pessoas, restaram apenas 1329, cerca de 15% do total. Na segunda tentativa, foram manualmente removidos do banco de dados os deslocamentos que se enquadravam no erro *arrivalTimeFormatException* ou no erro *departureTimeFormatException* e também foram corrigidos manualmente os erros de *neighbourhoodException*. Com o tratamento, o número de entradas caiu para 7908 pessoas, 93% do total. A transcrição resultou em 1745 pessoas, uma melhora de 31%. Apesar da melhora, a perda de informações ainda era relevante e acabava prejudicando a confiabilidade estatística da OD, pois essa já é feita com base em uma amostra da população. A solução foi tratar as principais inconsistências e refazer a transcrição. O procedimento realizado para o tratamento dessas inconsistências está detalhado no Anexo B. A partir disso, gerou-se três ODs transcritas:

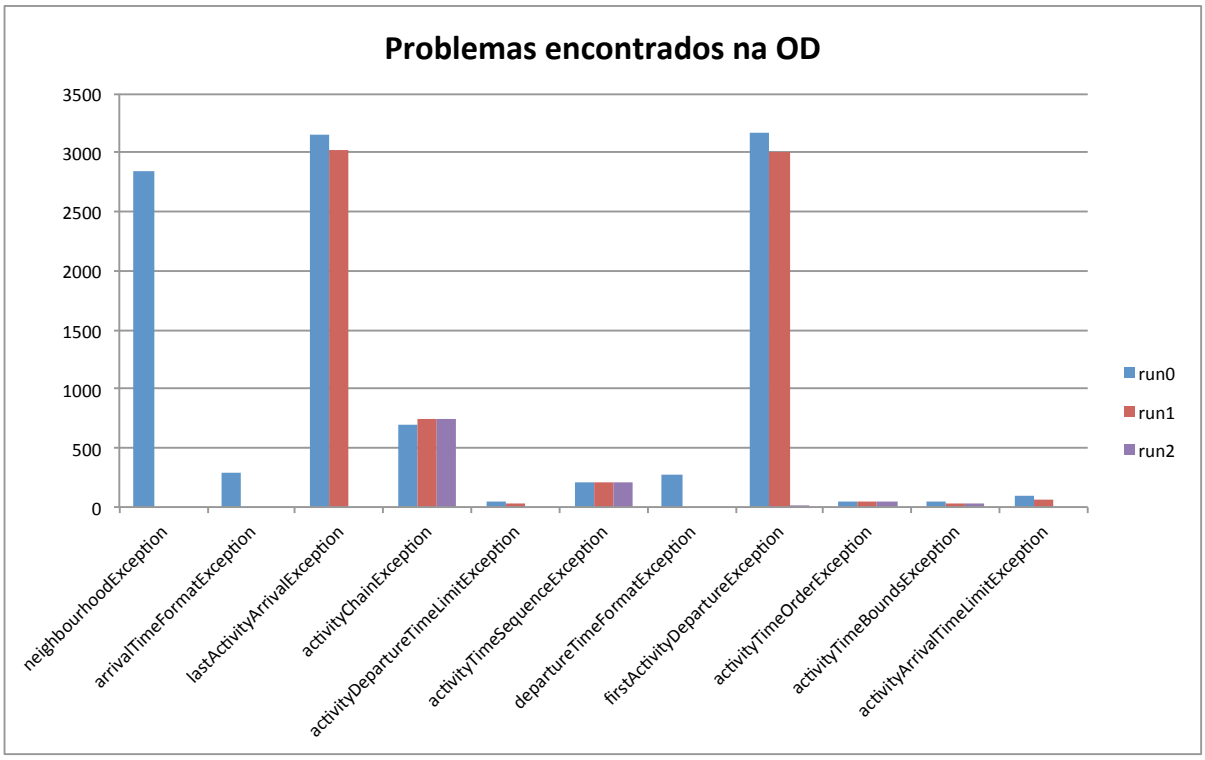
a) *run0* - banco de dados original da OD;

b) *run1* - banco de dados com a primeira correção;

c) *run2* - banco de dados corrigido e com tratamento de inconsistências.

A Figura 22 ilustra o resultado das três transcrições em termos de número de inconsistências.

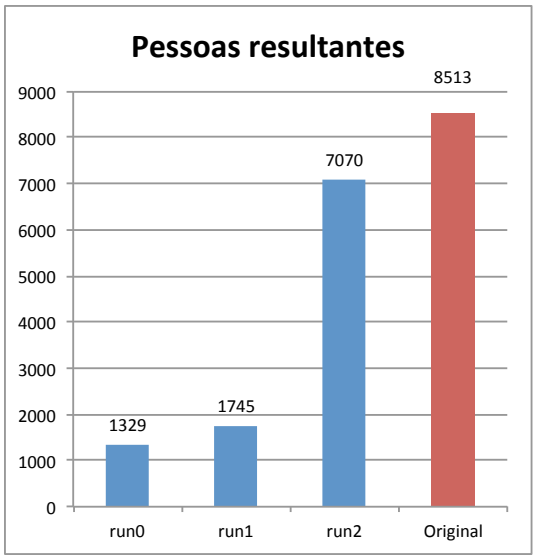
Figura 22 – Inconsistências nas transcrições.



Fonte: Elaborado pelo autor.

A principal melhora foi nas inconsistências *lastActivityArrivalException* e *firstActivityDepartureException* caindo para 0 e 22 ocorrências respectivamente. Como resultado, houve um aumento expressivo no número de pessoas transcritas, chegando a 7070, 83% do montante original. O aumento na quantidade de exceções do tipo *activityChainException* após o tratamento inicial ocorreu porque as pessoas que antes eram descartadas pelas exceções *arrivalTimeFormatException* e *departureTimeFormatException* não chegavam a ter a exceção *activityChainException* avaliada por impossibilidade. A partir do tratamento inicial isso se tornou possível, e a consequência foi o aparecimento das novas exceções. A Figura 23 ilustra as quantidades após o tratamento *run2*.

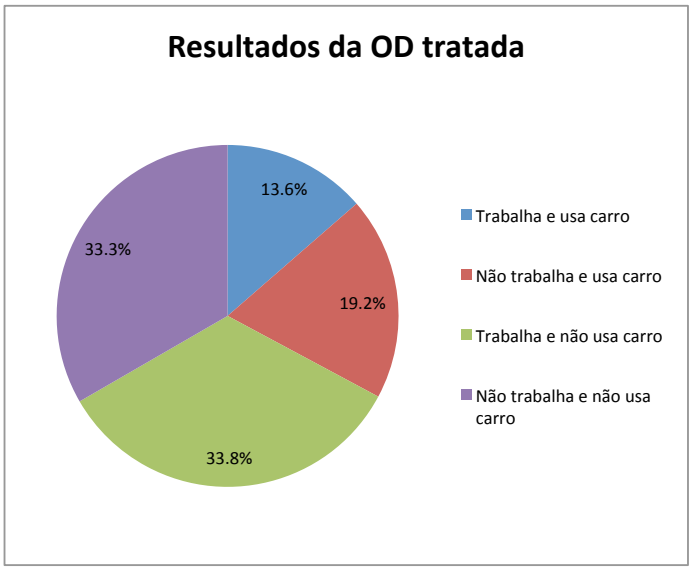
Figura 23 – Resultados nas diferentes transcrições e a quantidade original.



Fonte: Elaborado pelo autor.

Uma medida importante que depois é utilizada em outras etapas da modelagem é a proporção de pessoas com uso de carro em sua rotina e trabalham. A Figura 24 ilustra essa proporção.

Figura 24 – Proporção de pessoas com uso de carro e *status* de trabalho.



Fonte: Elaborado pelo autor.

Tabela 6 – Amostra de dados da OD transcrita resultante.

id	**activity**	**duration**	**mode**
101116104	home	17400	-1
101116104	work	35100.0	car
101116104	home	12300.0	car
101116104	education	13200.0	car
101116104	home	-1	car
101119104	home	54000	-1
101119104	shop	600.0	car
101119104	home	-1	car

Fonte: Elaborado pelo autor

A Tabela 6 apresenta uma amostra do arquivo da OD transcrita resultante. Ela mostra a rotina completa de duas pessoas. As atividades do dia iniciam em casa e terminam em casa. O valor -1 no modo da primeira atividade do dia significa que por ser a primeiro não há modo que levou a pessoa até ela. Já o valor -1 na duração da última significa que por ser a última atividade do dia a duração segue até o dia seguinte. Uma alteração importante é de que o valor correspondente à duração da primeira atividade do dia na verdade representa o horário de saída de casa. Essa mudança foi feita para se preservar os horários da rotina e ao mesmo tempo trabalhando com as durações para as outras atividades.

Quanto ao significado do *id*, seus três primeiros dígitos representam o bairro de origem, os três seguintes o número do cadastro, o próximo o número do morador e os dois últimos a posição na família (como descrito na Tabela 1, linha 5).

O procedimento foi executado numa rotina programada na linguagem Python e se encontra no Anexo B.

3.4.3 Estabelecimentos

O complemento gerado a partir dos dados de uso do solo, os estabelecimentos, refere-se às informações dos lotes da cidade e quais tipos de atividades são executadas neles. O MATSim usa esse tipo de informação para melhor alocar as atividades. Através de uma ferramenta fornecida no tutorial do MATSim, esses dados são convertidos em estabelecimentos distribuídos pela cidade de acordo com as coordenadas dos lotes. Em cada um dos estabelecimentos são permitidas determinadas atividades, tem um determinado horário de funcionamento e capacidade máxima de agentes executando atividades simultaneamente.

O arquivo com os lotes foi traduzido para um novo arquivo antes de ser passado na ferramenta de criação de estabelecimentos do MATSim. Os tipos de uso foram traduzidos segundo a Tabela 7.

Tabela 7 – Regra de transcrição dos diferentes usos de solo nos tipos de uso do MATSim.

Uso original	**Novo uso**
residencial	“home”
comercial	“work”, “shop”
industrial	“work”
serviços	“work”, “shop”
saúde	“work”, “shop”
ensino	“work”, “education”
cultural	“work”, “leisure”
’ religioso	“leisure”
financeiro	“work”, “shop”
institucional	“work”, “leisure”
rural	“work”
baldio	nenhum uso

Fonte: Elaborado pelo autor

A seguir foi inserido nos lotes as universidades. Uma lista das principais Instituições de Ensino Superior (IES) da cidade com suas respectivas localizações foi determinada. Com o auxílio do *software* Quantum GIS (QGIS, 2015), localizou-se os lotes mais próximos das IESs e posteriormente inseridas manualmente as atividades de universidade nos respectivos lotes. Dessa forma aquele lote, além de seus usos determinados pelos dados originais, teve incluída a atividade universidade.

Para a tradução do arquivo gerado no complemento final “Estabelecimentos”, foi usada ferramenta própria do MATSim (Anexo E.4), informando o horário de funcionamento desejado e capacidade de execução simultânea para cada tipo de atividade. A Tabela 8 mostra os valores utilizados para os diferentes tipos de estabelecimentos. Nela, as capacidades levam em conta a simplificação de somente modelar agentes com carro.

Tabela 8 – Valores considerados para horário de funcionamento e capacidade dos estabelecimentos de acordo com o uso.

Tipo de uso	**Horário de funcionamento**	**Capacidade**
shop	das 8h às 19h	10
leisure	das 8h às 21h	30
work	das 5h às 23h	20
education	das 7h30 às 22h30	100
university	das 7h30 às 22h30	1000
home	24h	3

Fonte: Elaborado pelo autor

Uma amostra do arquivo resultante é mostrado na Tabela 9.

Tabela 9 – Amostra de dados dos estabelecimentos resultantes.

id	**Coord_X**	**Coord_Y**	**Types**
13340	715458.183787	7082353.195955	work,shop
13341	715640.521099	7082284.735297	home
13342	715650.626984	7082273.601658	home,work,shop
13343	715629.222585	7082261.447016	home,work,shop
13344	715619.158326	7082250.897469	home
13345	715608.850256	7082240.494159	home

Fonte: Elaborado pelo autor

Os três primeiros dígitos do *id* representam o bairro onde se localiza e os demais seu identificador único.

O procedimento foi executado numa rotina programada na linguagem Python e se encontra no Anexo C.

3.4.4 Confecção da população

A partir dos dados do censo sintético, da OD transcrita e dos estabelecimentos, além da rede viária, é possível a confecção da população de agentes. Para esta parte do processo a ferramenta é fornecida no tutorial do MATSim, porém algumas modificações foram feitas. A ferramenta compreende três *classes* programadas na linguagem Java. A primeira classe executada é a *CreatePopulation*, responsável pela criação da população de agentes. A segunda a ser executada é a classe *CreateDemand*, responsável pela criação dos planos e alocação desses planos nos agentes. Por último, a classe *CreatePopulationAndDemand*, realiza a coordenação das duas anteriores e transcrição do resultado no formato do MATSim. As classes foram adaptadas e encontram-se no Anexo E.

Os passos que a ferramenta segue são:

a) para cada pessoa do censo sintético é sorteado uma pessoa na OD de acordo com *status* de trabalho;

b) a rotina da pessoa da OD é adaptada à pessoa do censo:

   – sorteia-se um *estabelecimento* de moradia e de trabalho (caso a pessoa trabalhe) de acordo com os bairros de moradia e trabalho da pessoa do censo;

   – para as outras atividades da rotina sorteia-se um *estabelecimento* no entorno daquele de sua última atividade;

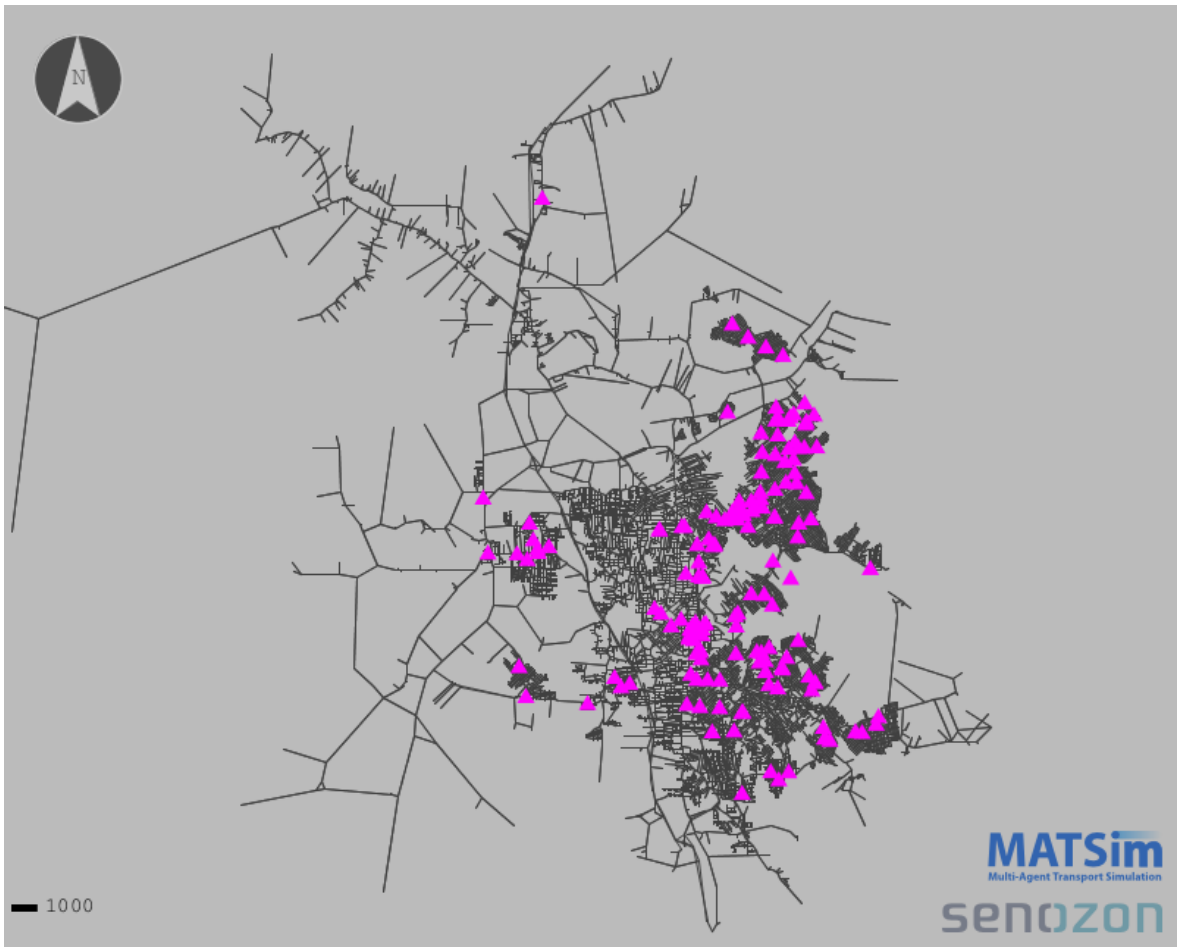
c) os horários de início de fim das atividades são variados seguindo uma curva normal;

d) as pessoas formadas são então escritas na população final.

3.5 Modelo construído

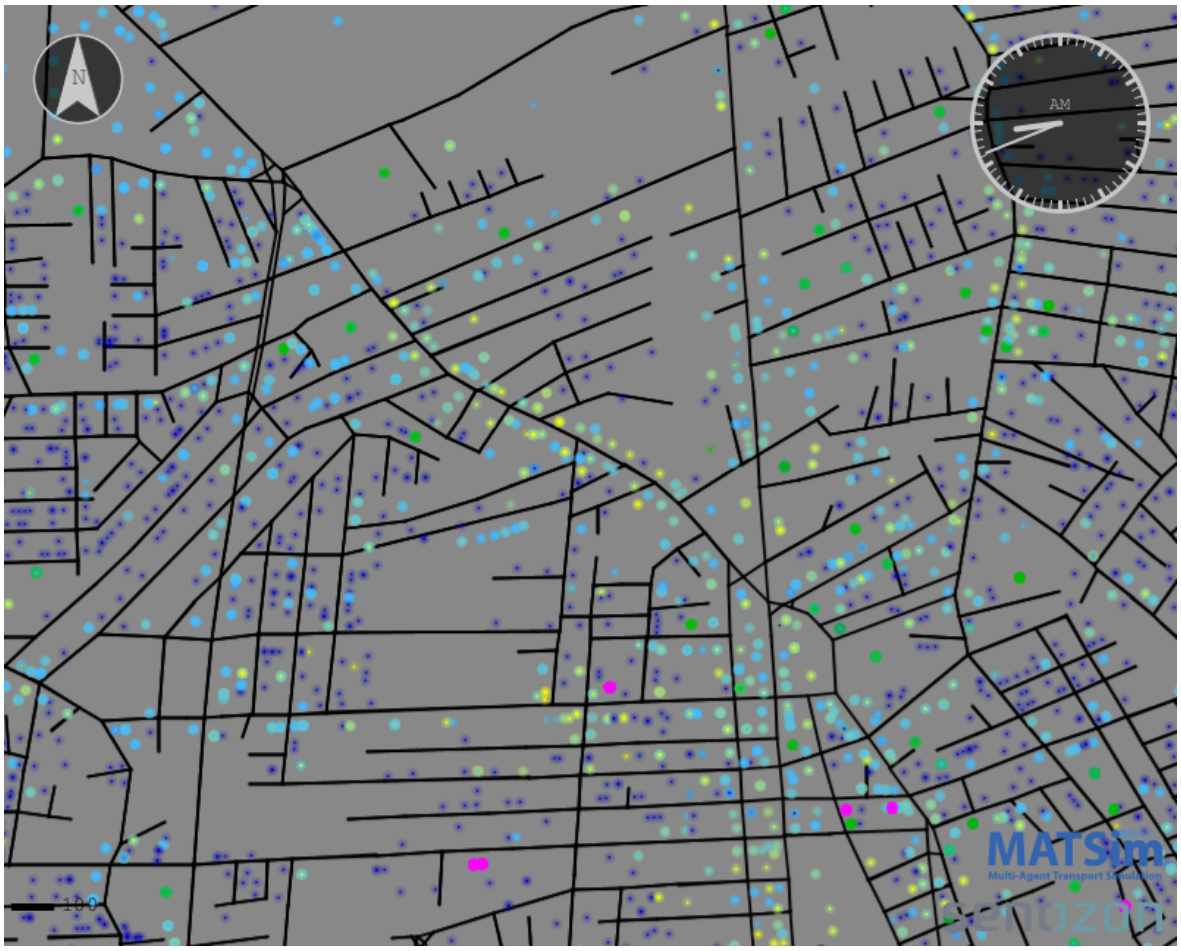
O modelo, ao fim do processo de construção, resultou em cerca de 32 mil arcos, 163 mil agentes e 115 mil estabelecimentos. Para exemplificar, a Figura 25 ilustra a disposição dos estabelecimentos de educação básica no modelo, que é onde o agentes de menos de 18 anos realizam a atividade "education". Por outro lado, a Figura 26 ilustra, como exemplo, as atividades ocorrendo no bairro Santo Antônio às 16h (sem a otimização do simulador). Nela, os pontos em rosa são atividades de educação, os pontos em azul-escuro de residência, os pontos em azul-claro de trabalho, os pontos em amarelo de compras e os pontos em verde de lazer.

Figura 25 – Localização dos estabelecimentos do tipo "education" no modelo de Joinville.



Fonte: Elaborado pelo autor.

Figura 26 – Atividades sendo realizadas no bairro Santo Antônio às 16h.



Fonte: Elaborado pelo autor.

3.5.1 Limitações do modelo

Por conta das considerações feitas na seção 3.4, das características dos dados de entrada utilizados e da configuração básica do MATSim, o modelo construído contará com algumas limitações. Dentre elas, as principais são:

a) apenas viagens utilizando carros foram modeladas, excluindo transporte coletivo, caminhada, bicicleta, motocicleta, etc;

b) falta de semáforos

c) a rede não conta com impedâncias como lombadas e faixas de pedestre;

d) diferenças no pavimento desconsideradas;

e) o modelo deixa de fora incidentes de tráfego ou obras, que modificariam as vias;

4 Resultados e discussões

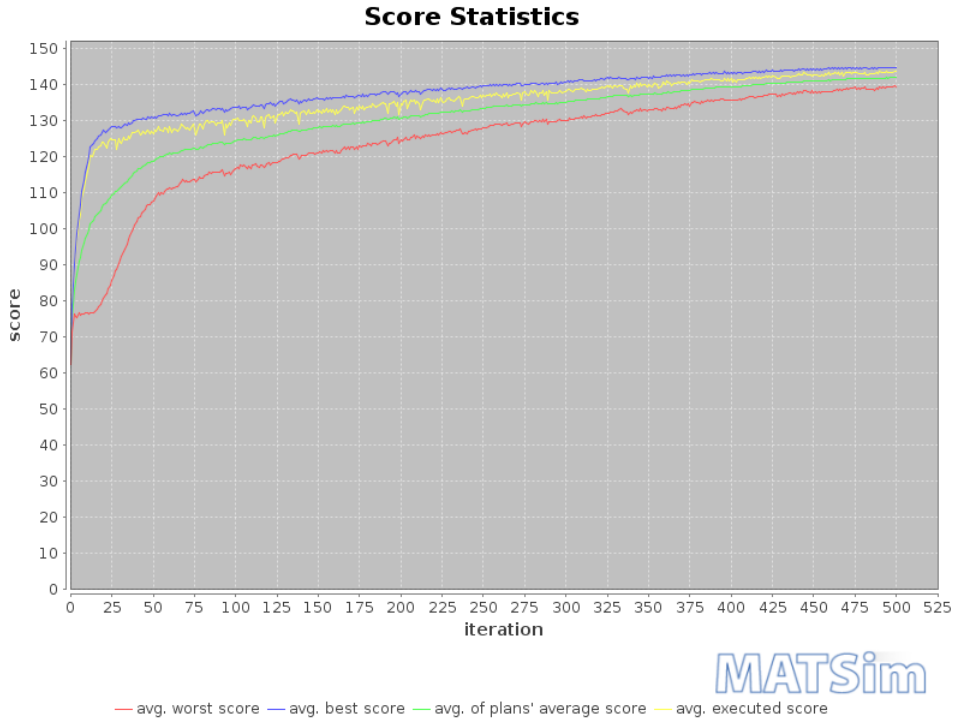
Com a construção do modelo de Joinville, pode-se realizar a simulação com o MATSim. Neste capítulo são apresentados os parâmetros de configuração da simulação, os principais resultados dessa simulação, a avaliação do modelo em comparação com as contagens de tráfego reais e algumas análises agregadas possíveis.

4.1 Configuração da simulação

Os parâmetros de simulação utilizados seguiram o padrão do MATSim, em sua versão 0.6.1, e para chegar a um estado de equilíbrio foram necessárias 350 iterações. Neste trabalho foi utilizado o critério de parada de 500 iterações.

A Figura 27 ilustra a evolução das notas dos planos da população ao longo das iterações. A linha vermelha representa a média das piores notas dos planos, a azul a média das melhores notas, a verde a média da média das notas dos planos e a linha amarelo a média das notas dos planos executados pelos agentes.

Figura 27 – Evolução das notas avaliadas pelo MATSim ao longo das iterações.



Fonte: Elaborado pelo autor.

Utilizando 2GB de memória, a simulação demorou cerca de 36 horas para concluir

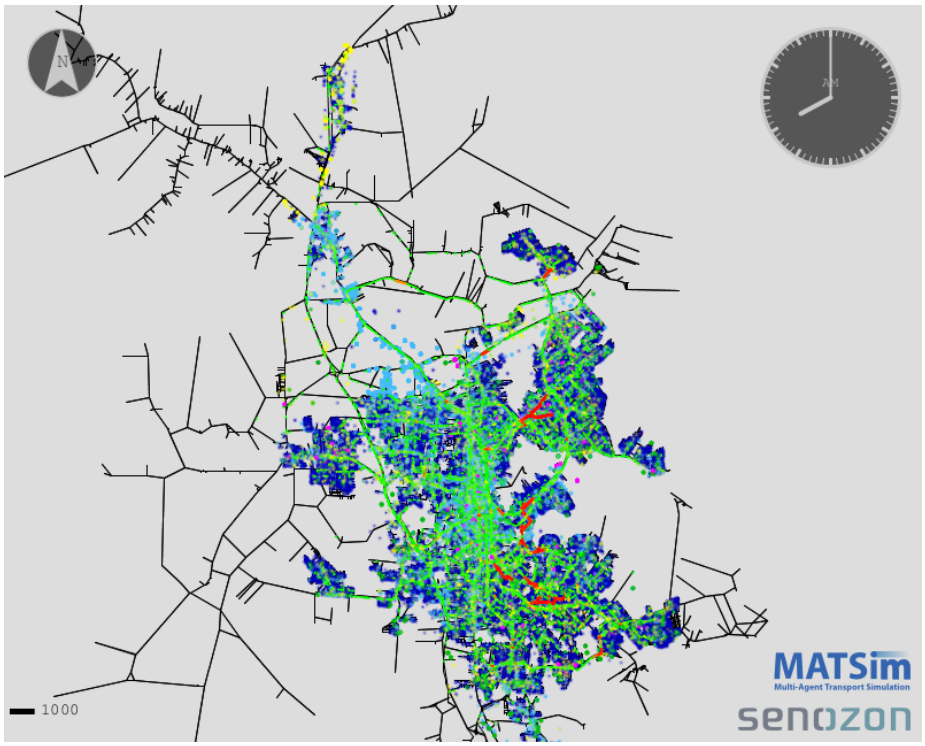
as quinhentas iterações. Quanto ao espaço em disco, escrevendo o resultado apenas a cada 100 iterações, ocupou 20GB de memória física.

4.2 Principais resultados da simulação

O MATSim oferece seus resultados no formato de *eventos*, que identifica tudo que um agente faz ao longo de sua rotina. Os eventos podem ser sair de casa, entrar e sair de um arco, iniciar ou terminar uma atividade, etc. Dessa forma, o resultado da simulação é genérico e desagregado e, para analisá-lo é necessário fazer sua agregação. Para isso, foi utilizada a ferramenta de análise e visualização de simulações do MATSim denominada Via (SENOZON AG, 2015).

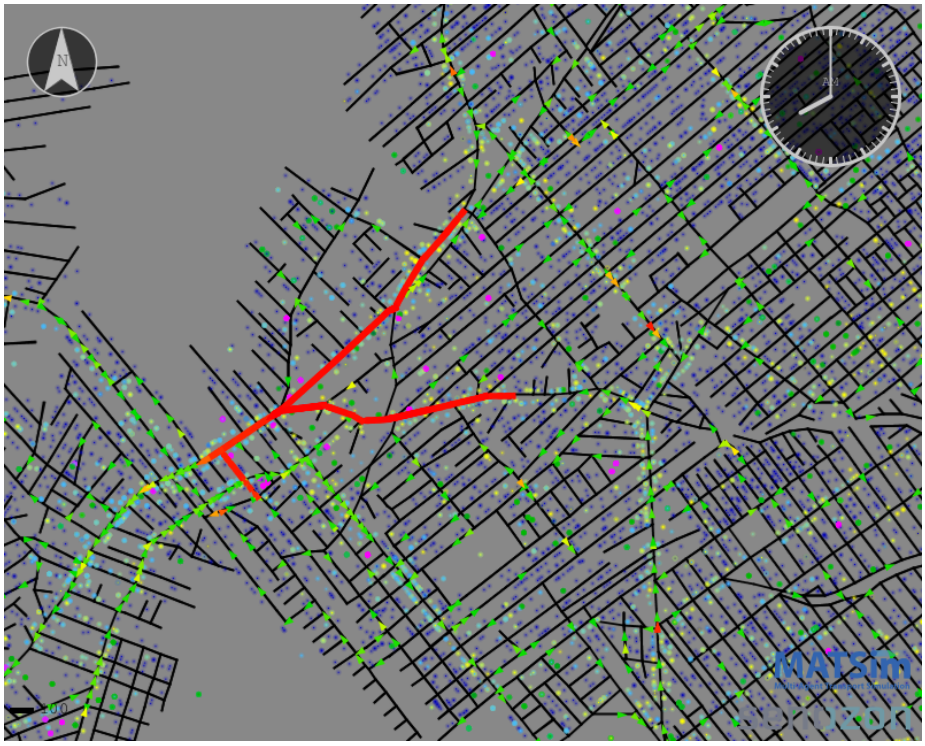
A Figura 28 ilustra a visualização do resultado final da simulação no Via. Os pontos em verde ou vermelho ao longo das vias são os agentes se locomovendo ao longo dos arcos da rede. A cor verde significa que os agentes estão em velocidade de fluxo livre ou próximo disso e os agentes em vermelho estão parados ou quase parados, cores em amarelo ou laranja são de velocidades intermediárias. Os pontos coloridos fora da rede viária representam os agentes executando alguma atividade em um estabelecimento (as cores são as mesmas das apresentadas na Figura 26). A Figura 29 ilustra o resultado para o bairro Iririú às 8h, onde destaca-se os congestionamentos nas principais vias de saída do bairro.

Figura 28 – Visualização do resultado da simulação do modelo de Joinville.



Fonte: Elaborado pelo autor.

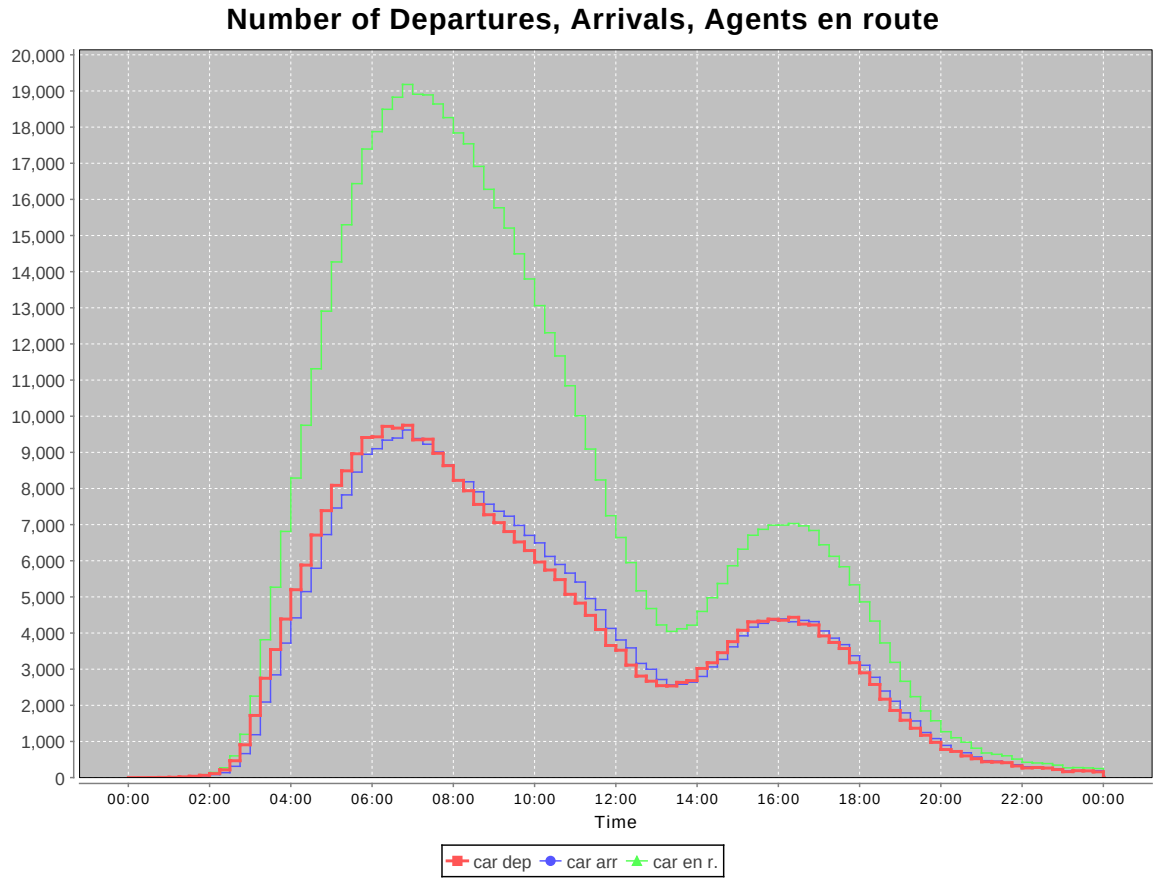
Figura 29 – Visualização do resultado da simulação para o bairro Iririú.



Fonte: Elaborado pelo autor.

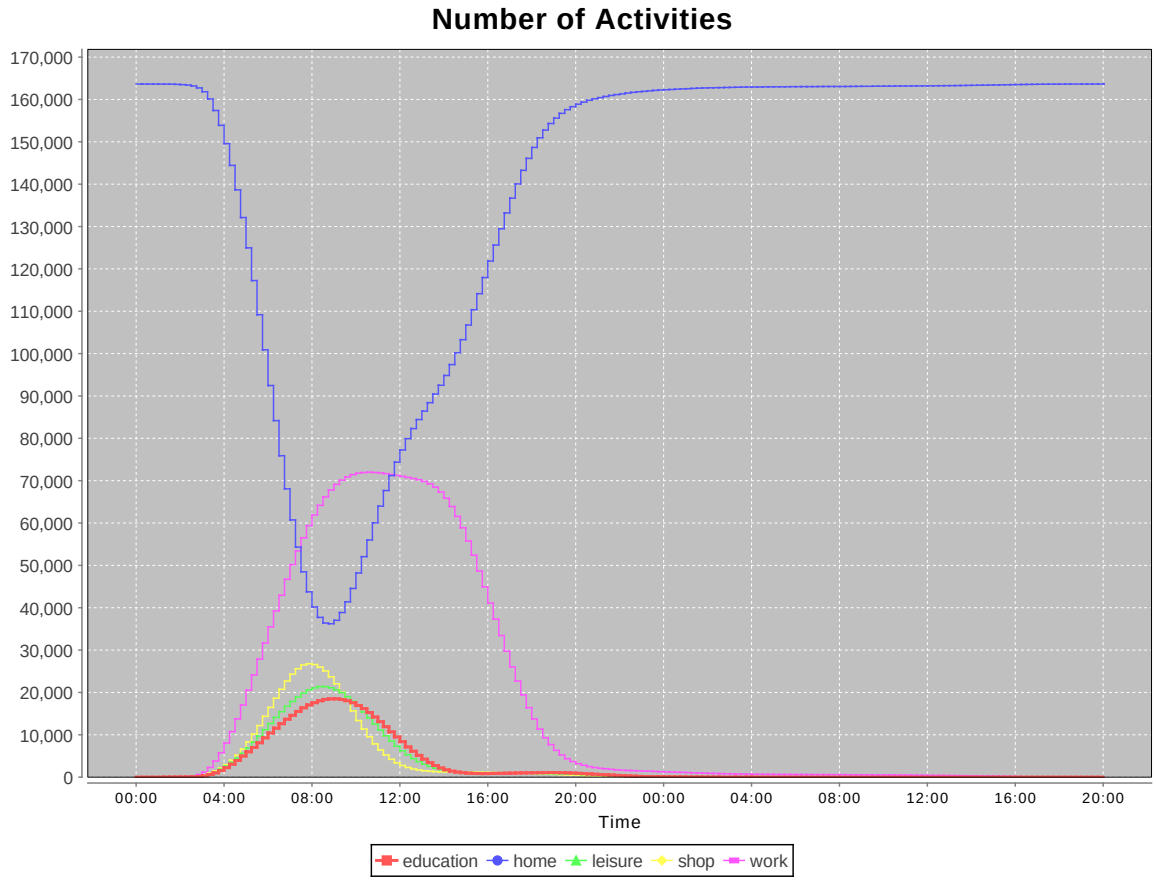
A Figura 30 e a Figura 31 ilustram dados estatísticos da simulação. Nelas, o eixo vertical representa as quantidades enquanto o eixo horizontal representa o tempo. Na Figura 30 pode se visualizar a evolução dos deslocamentos ao longo do dia. Destaca-se os dois picos, que representam os picos de tráfego no modelo, pela manhã e pela tarde. Já na Figura 31 pode se visualizar o número de atividades sendo executadas de acordo com seu tipo e horário do dia. Na linha rosa, destaca-se o horário de trabalho dos agentes.

Figura 30 – Número de viagens no tempo.



Fonte: Elaborado pelo autor.

Figura 31 – Número de atividades no tempo.



Fonte: Elaborado pelo autor.

4.3 Avaliação dos resultados

A avaliação dos resultados da simulação foi feita com base nos dados de contagens fornecidos (seção 3.1.5). A vazão de tráfego nos arcos que se tinha os dados reais foram agregados por hora e comparados aos valores de contagens reais. Para ilustrar, foram selecionados os gráficos comparativos para quatro arcos. Por verificação empírica, escolheu-se dois dos melhores e dois dos piores. Os quatro arcos selecionados são apresentados na Figura 32, sendo os arcos em verde os que obtiveram um bom resultado e os em vermelho os que obtiveram um resultado ruim. Os arcos escolhidos pertencem às vias:

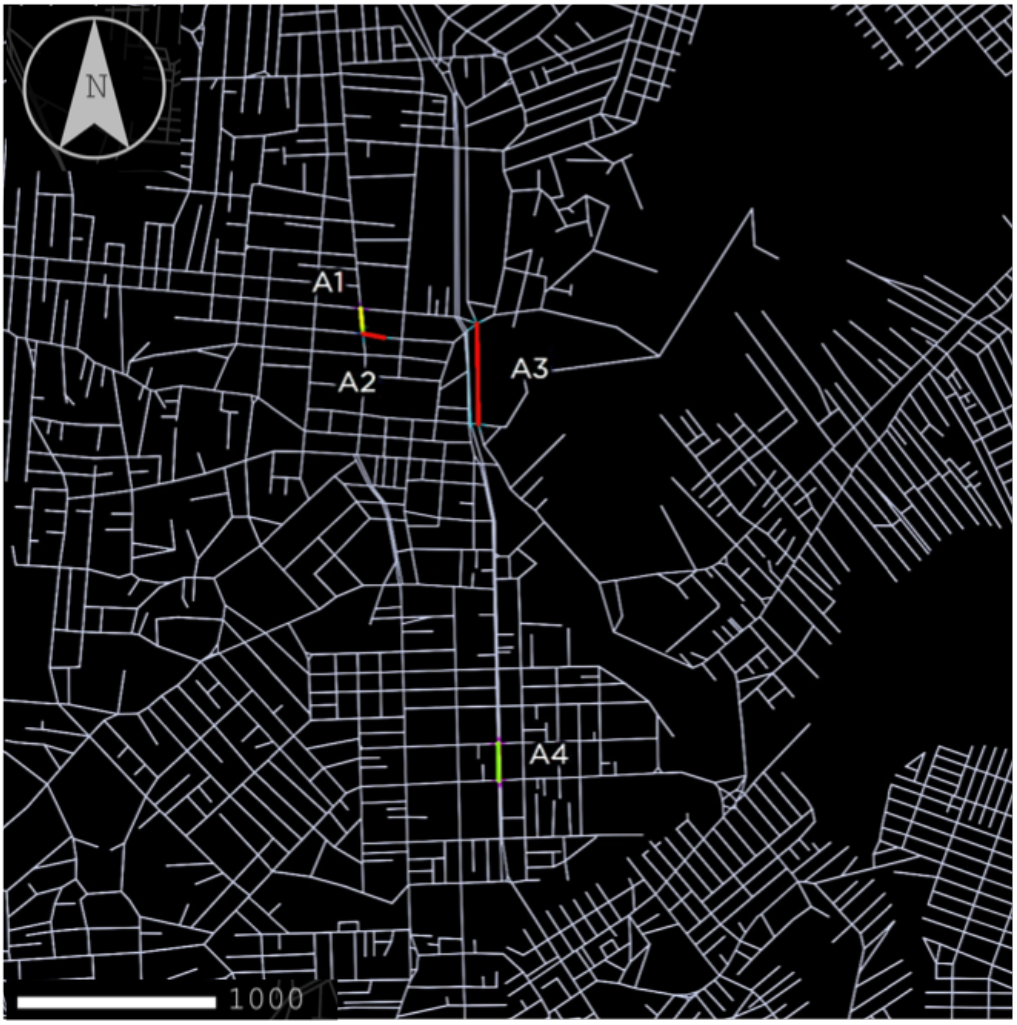
a) Rua João Colin (A1);

b) Rua Tijucas (A2);

c) Avenida Hermann August Lepper (A3); e

d) Avenida Coronel Procópio Gomes de Oliveira (A4).

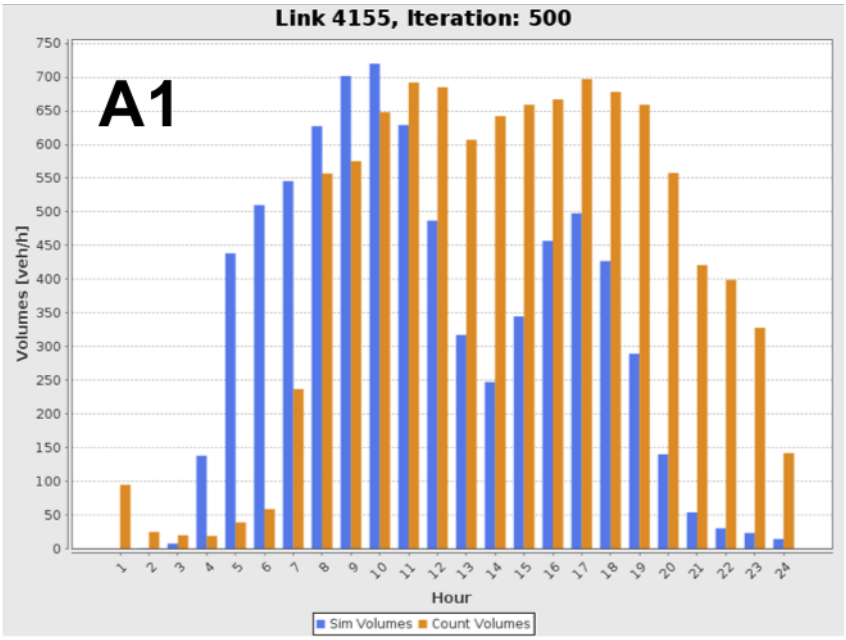
Figura 32 – Arcos selecionados para ilustração dos resultados das contagens.



Fonte: Elaborado pelo autor.

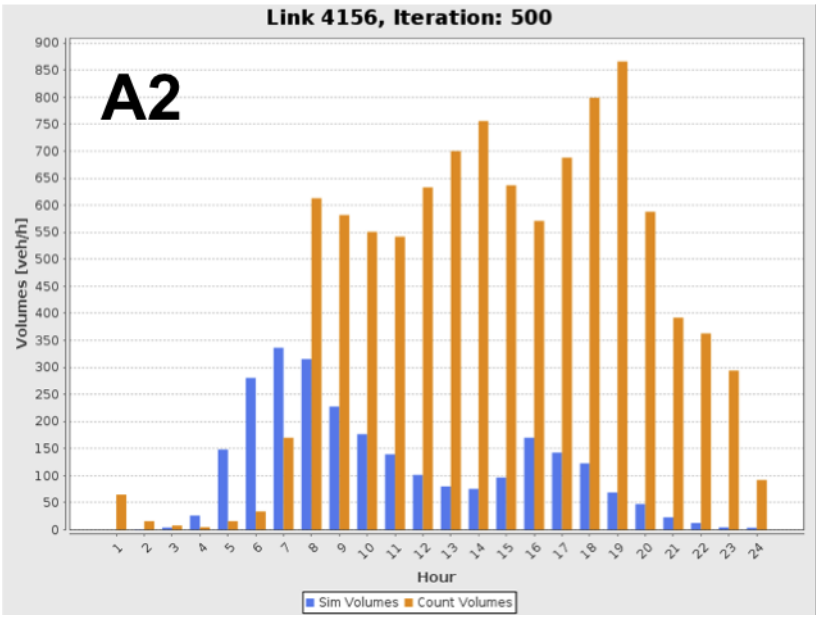
Os valores dos arcos de contagem respectivos são apresentados nas Figuras 33, 34, 35 e 36.

Figura 33 – Resultado comparativo no arco A1, localizado na Rua João Colin.



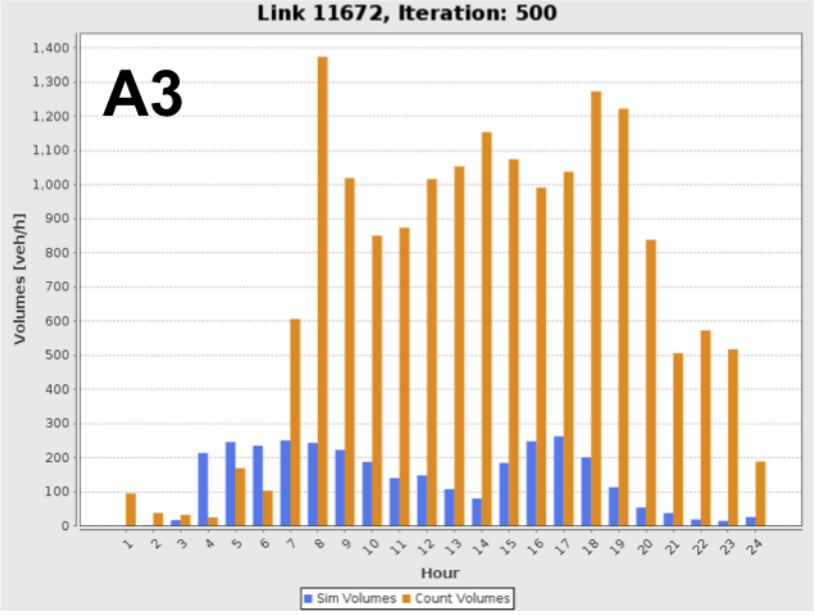
Fonte: Elaborado pelo autor.

Figura 34 – Resultado comparativo no arco A2, localizado na Rua Tijucas.



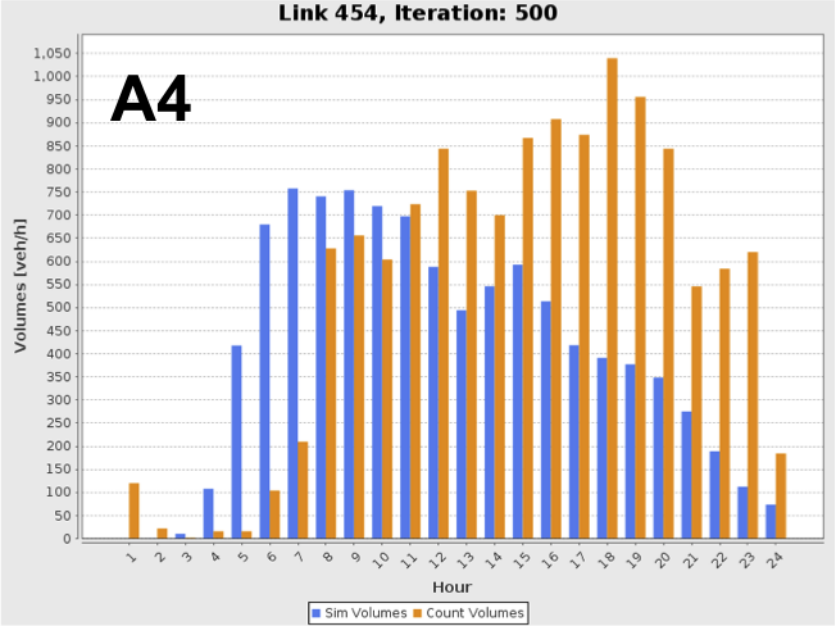
Fonte: Elaborado pelo autor.

Figura 35 – Resultado comparativo no arco A3, localizado na Avenida Hermann August Lepper.



Fonte: Elaborado pelo autor.

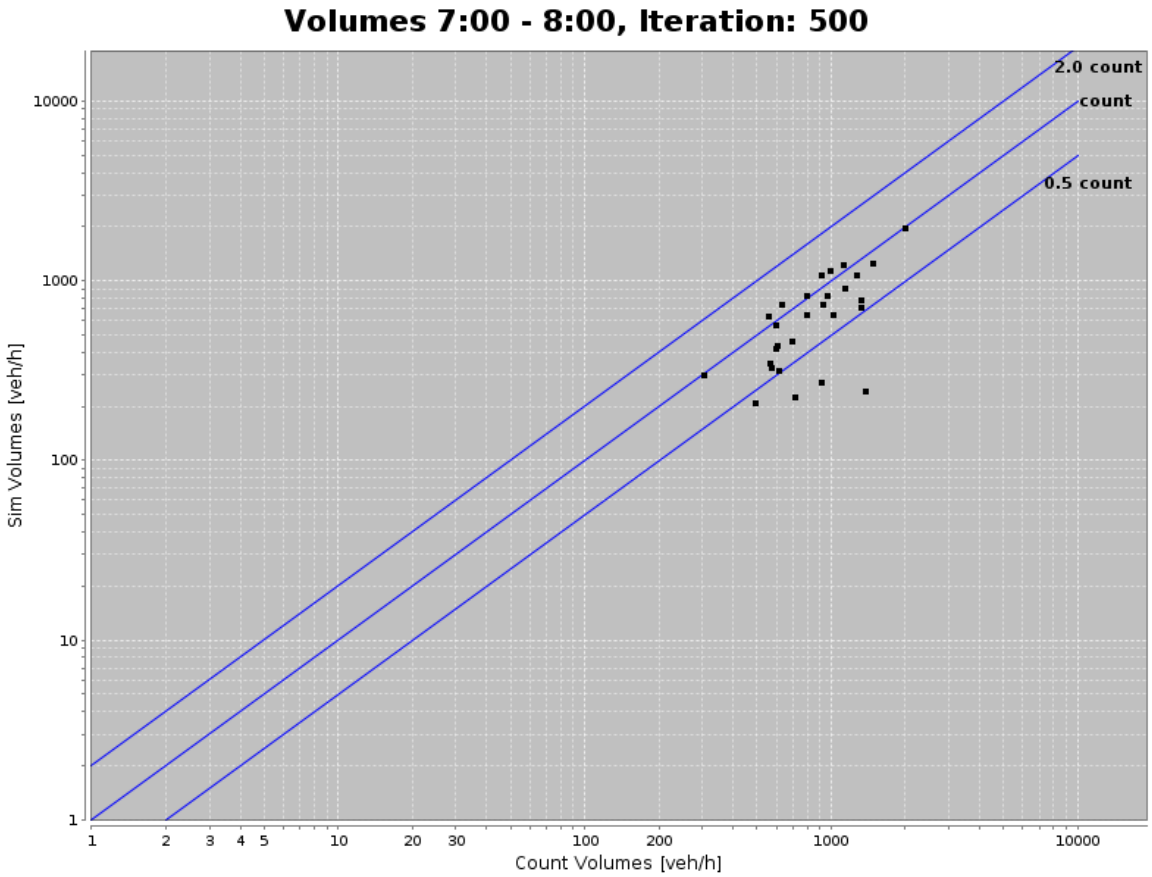
Figura 36 – Resultado comparativo no arco A4, localizado na Avenida Coronel Procópio Gomes de Oliveira.



Fonte: Elaborado pelo autor.

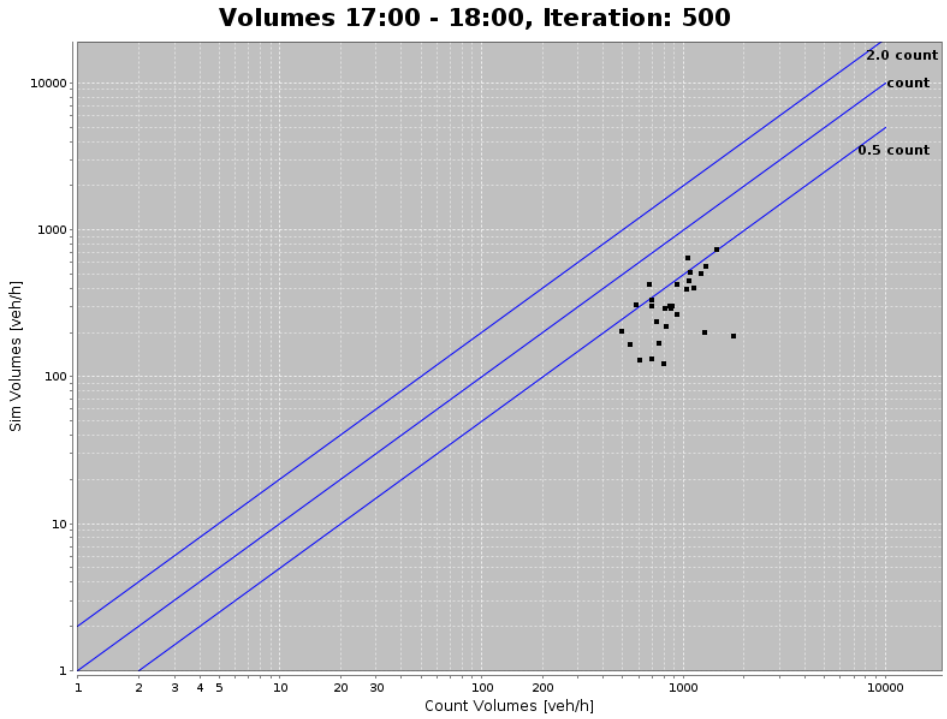
A Figura 37 apresenta a comparação dos resultados de todos os arcos de contagem para o pico da manhã, entre 7h e 8h, enquanto a Figura 38 faz a comparação para o pico da tarde, entre 17h e 18h. Já a Figura 39 mostra a comparação para todo o dia. Nessas figuras, as linhas horizontais representam as contagens, sendo a central equivalente a 100% do valor das contagens, a superior a 200% e a inferior a 50%.

Figura 37 – Comparação dos resultados da simulação no pico da manhã.



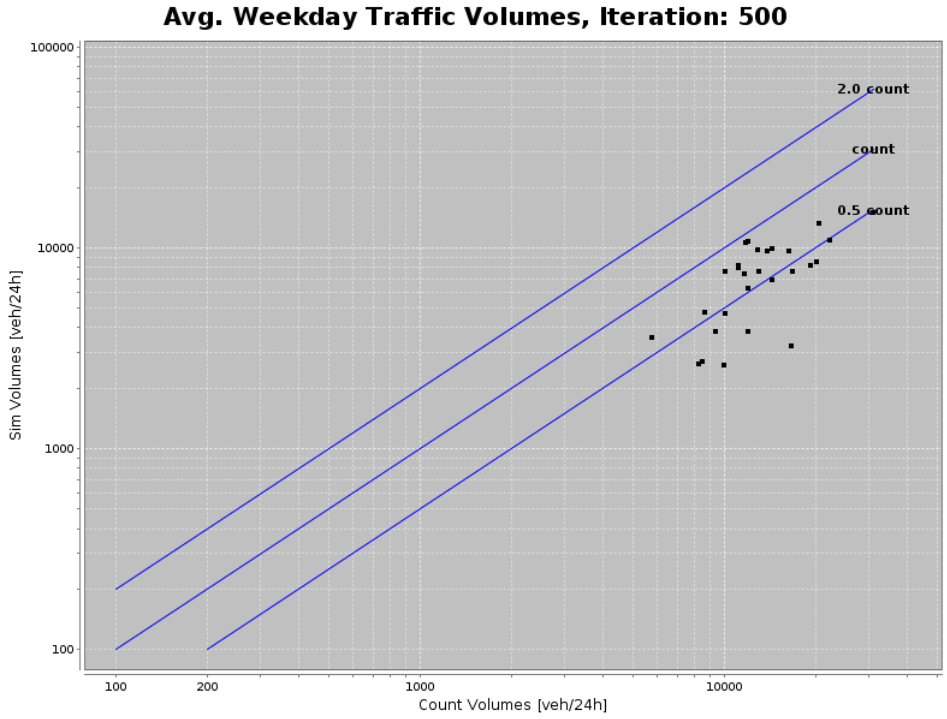
Fonte: Elaborado pelo autor.

Figura 38 – Comparação dos resultados da simulação no pico da tarde.



Fonte: Elaborado pelo autor.

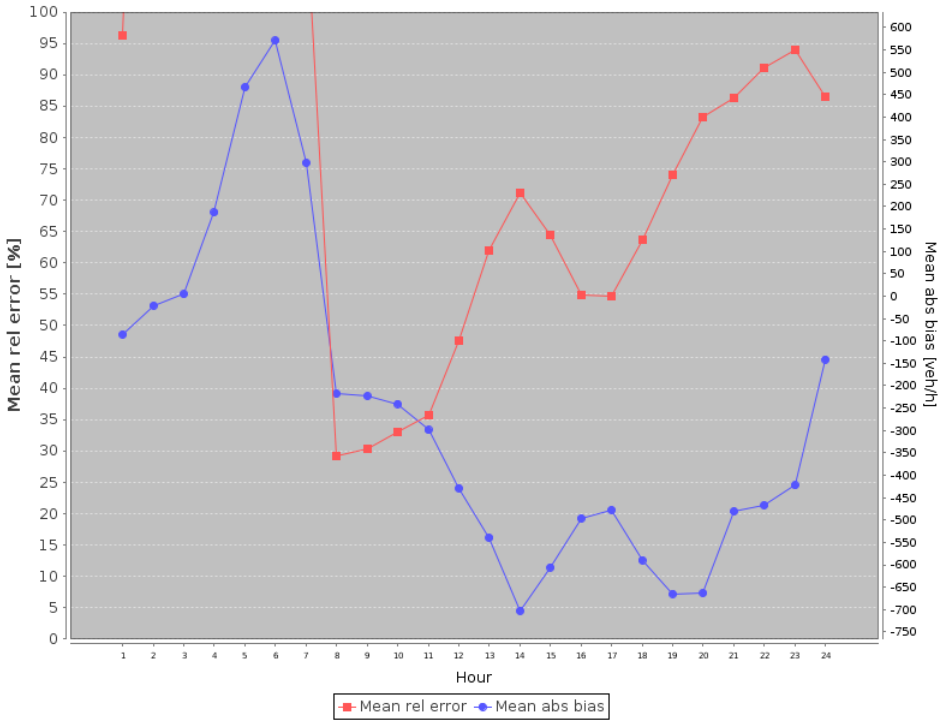
Figura 39 – comparação dos resultados da simulação para o dia inteiro.



Fonte: Elaborado pelo autor.

Fazendo uma avaliação dos erros absolutos e relativos dos dados simulação em relação aos valores das contagens, obtêm-se o gráfico da Figura 40. Na figura, é possível perceber que o melhor resultado foi no período entre 8h e 10h, com erros relativos próximos de 30%. O pico da tarde, entre 16h e 17h também apresentou erros relativos próximos a 55%. A figura ainda ilustra, a partir da curva de erros absolutos, em azul, que os valores simulados ficaram abaixo do valores reais para a maior parte do dia, a partir das 8h da manhã.

Figura 40 – Erro relativo e absoluto ao longo do dia para todos os arcos de contagem.



Fonte: Elaborado pelo autor.

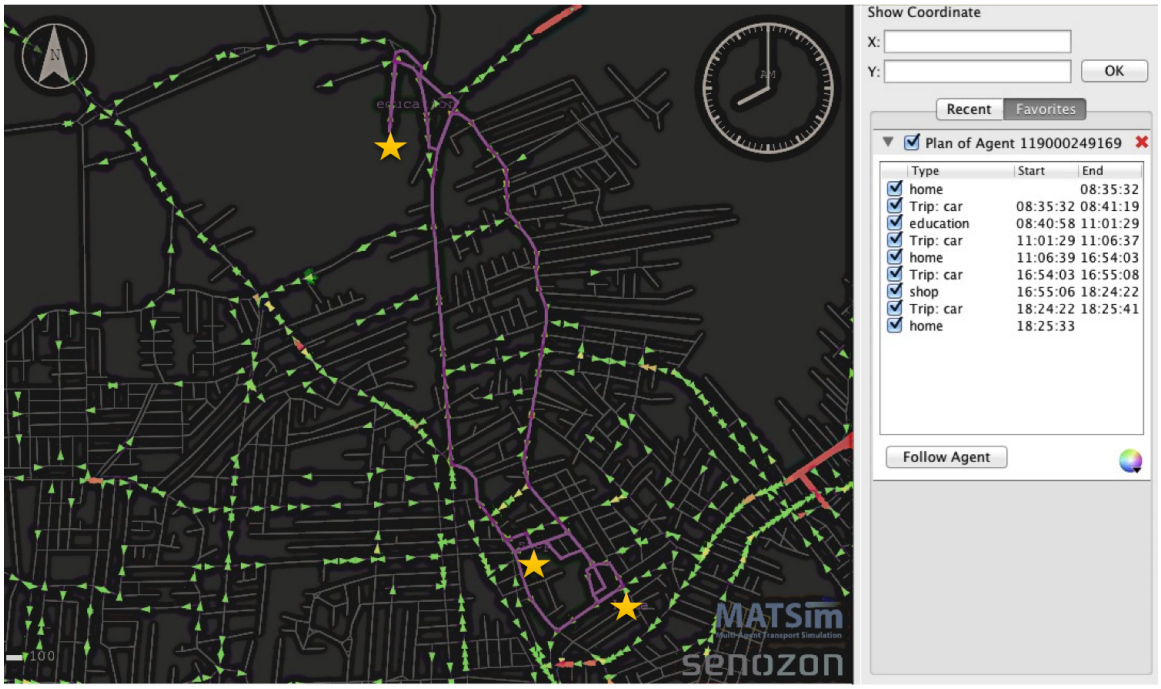
4.4 Análises agregadas

Os eventos não são o tipo de dado que se utilizaria para, por exemplo, a escolha de uma nova linha de ônibus ou da implantação de um novo zoneamento. Para uso efetivo da simulação no planejamento de transportes faz-se necessário a agregação desses resultados tornando possível a extração de informações relevantes. O *software* Via permite diferentes tipos de análise agregada dos dados do MATSim. Exemplos dessas análises serão apresentadas nesta seção.

A primeira análise possível é a observação do comportamento de agentes específicos. A Figura 41 ilustra um agente que vive no bairro Saguaçu, estuda na Zona Industrial

Norte e ao fim do dia faz compras próximo à sua casa. A tabela ao lado na figura descreve os horários de suas atividades e deslocamentos.

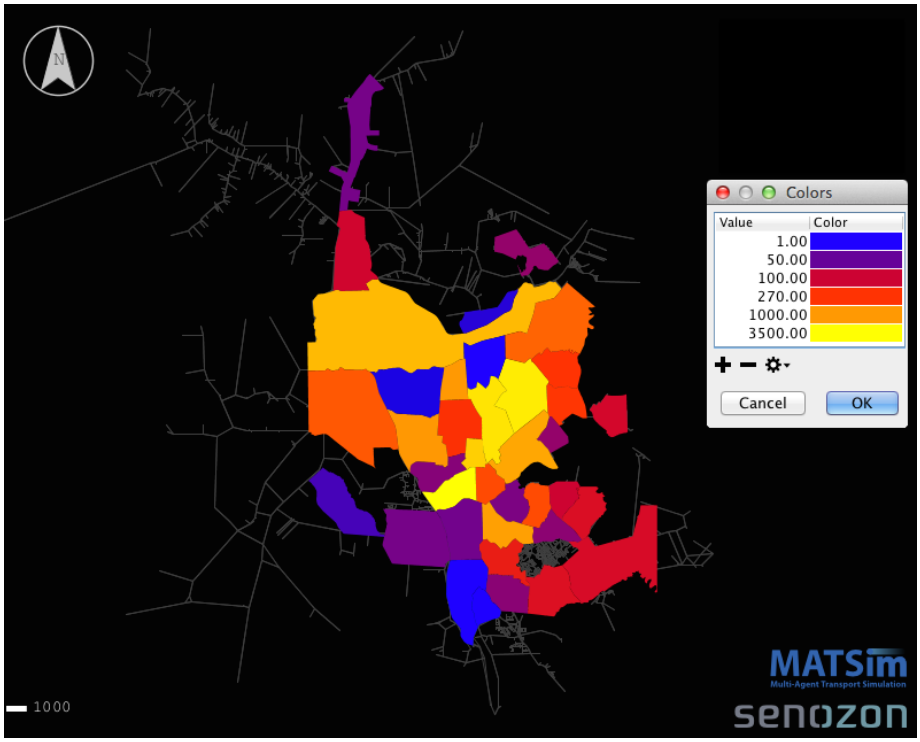
Figura 41 – Análise do plano de agentes.



Fonte: Elaborado pelo autor.

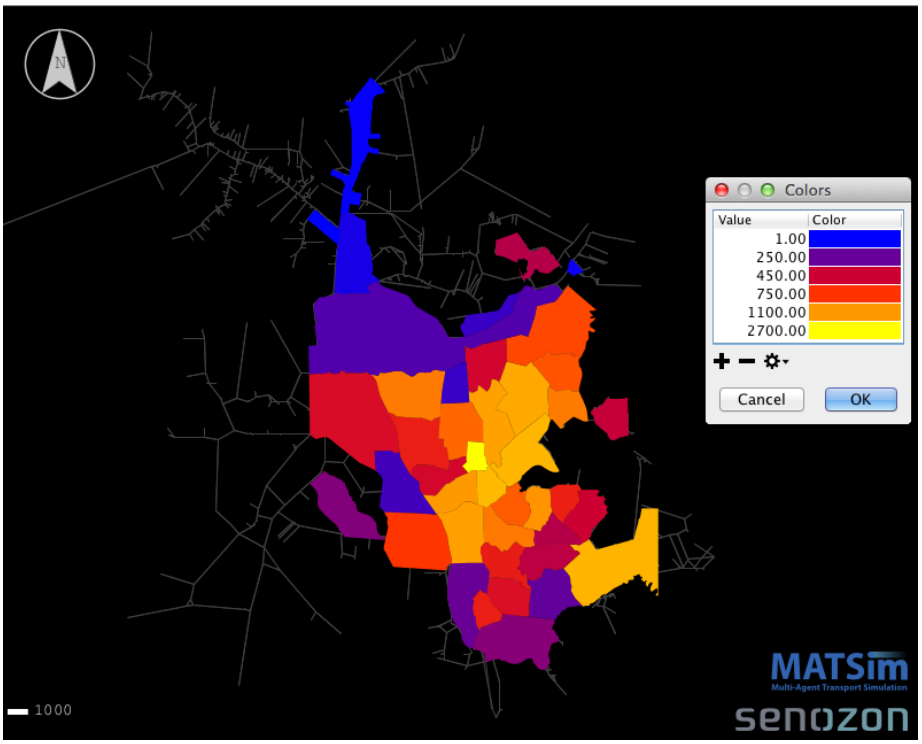
O Via permite também a agregação de atividades por bairro por período. As Figuras 42 e 43 ilustram essa agregação para um dia completo. Nelas observa-se a quantidade relativa de atividades dos tipos educação e lazer por bairro. A representação da intensidade das atividades dá-se pelas cores, que vão do azul ao amarelo passando pelo vermelho.

Figura 42 – Atividades do tipo educação por bairro.



Fonte: Elaborado pelo autor.

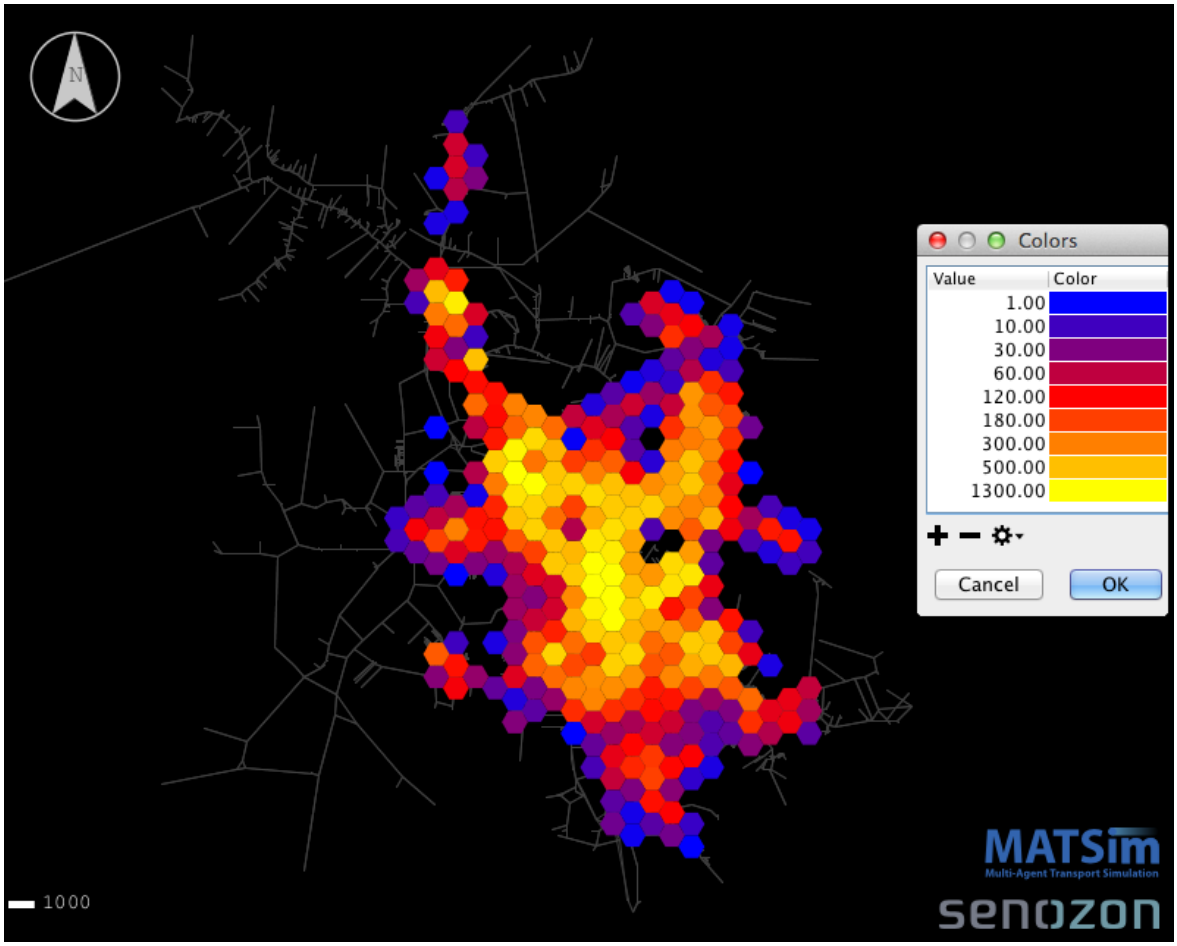
Figura 43 – Atividades do tipo lazer por bairro.



Fonte: Elaborado pelo autor.

A agregação também pode ser feita por seção hexagonal, como ilustra a Figura 44, para a atividade trabalho.

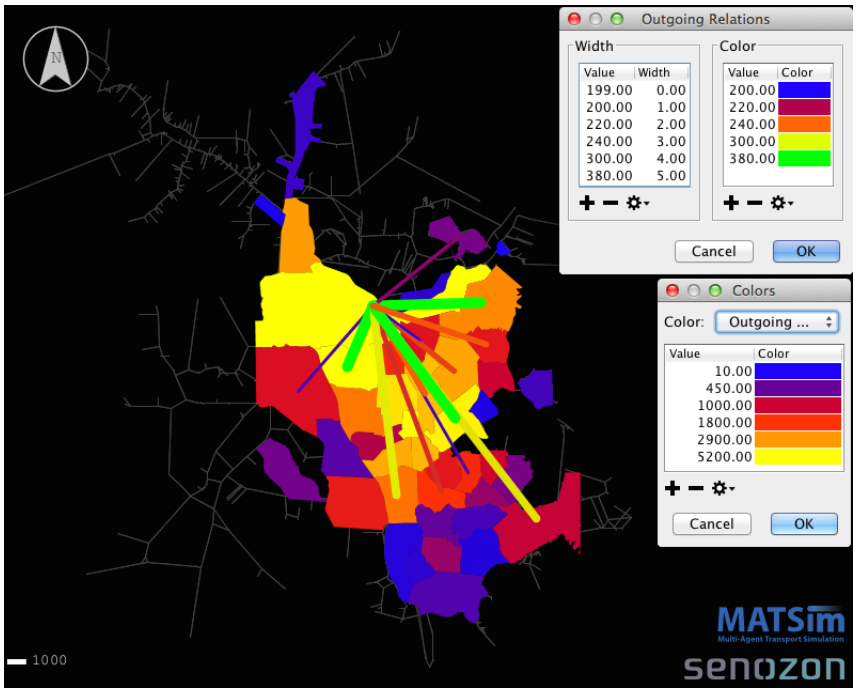
Figura 44 – Atividades do tipo trabalho por seção hexagonal.



Fonte: Elaborado pelo autor.

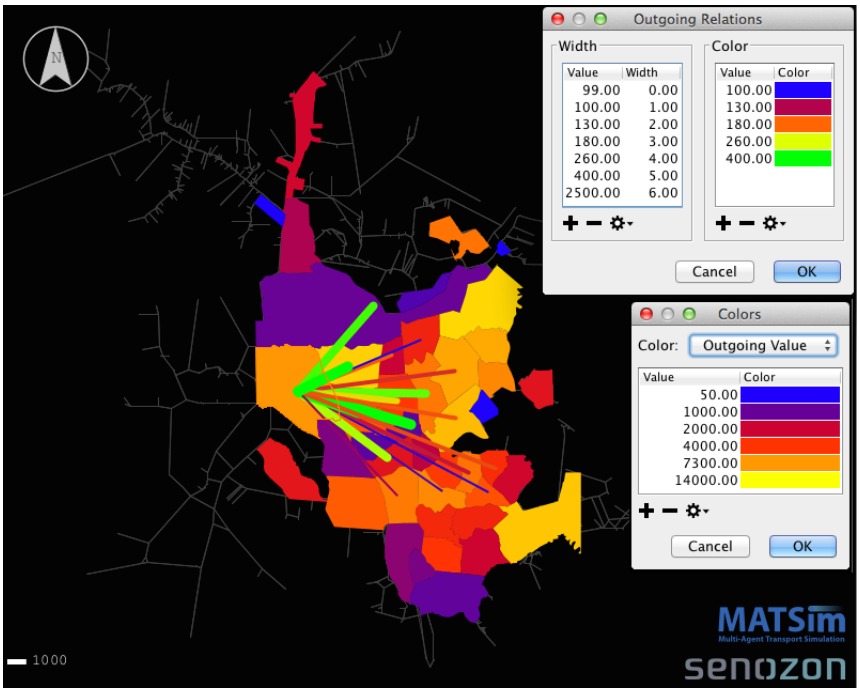
É possível também fazer uma análise composta do tipo origem-destino, conforme é ilustrado nas Figuras 45 e 46. Nas duas figuras, a coloração dos bairros é de acordo com o número de viagens de saída de todo o dia. Na primeira, analisa-se a quantidade de deslocamentos com destino trabalho que chegam ao bairro Zona Industrial Norte partindo dos outros bairros da cidade. A espessura das linhas entre os bairros representa a quantidade de viagens. Na segunda figura, analisa-se a quantidade de deslocamentos do tipo casa que saem do bairro Vila Nova em direção aos outros bairros da cidade. Na Figura 46, destacam-se as maiores quantidades para os bairros Costa e Silva, Glória, Centro e Zona Industrial Norte.

Figura 45 – Viagens do tipo trabalho que tem o bairro Zona Industrial Norte como destino.



Fonte: Elaborado pelo autor.

Figura 46 – Viagens do tipo casa que tem o bairro Vila Nova como origem.



Fonte: Elaborado pelo autor.

4.5 Discussões finais

O presente capítulo tratou de estudar os dados de saída da simulação do modelo construído no Capítulo 3. A configuração, mostrada na seção 4.1, segue os padrões básicos do MATSim e a simulação em si demonstrou demandar alto custo computacional.

O capítulo apresentou os resultados básicos da simulação, que são os eventos (seção 4.2). A partir destes eventos é que se pode tirar todas as outras análises agregadas. Os eventos puderam ser visualizados e analisados com o uso do *software* Via. A seção 4.4 mostrou algumas das diversas análises agregadas possíveis a partir dos dados resultantes da simulação.

Quanto à avaliação do modelo, a seção 4.3 apresentou a comparação das vazões de tráfego do modelo em relação a alguns arcos de contagem na cidade e ainda comparações gerais agregadas. Ficou ilustrado que houve uma boa comparação nos horários de pico mas que no geral o modelo se mostrou subdimensionado.

A partir dos exemplos de análises apresentadas, pôde-se verificar a diversidade de informações que podem ser obtidas a partir do resultado da simulação usando o MATSim. Dentro do contexto de planejamento de transportes, esse tipo de resultado pode ser relevante em prover os mais diversos tipos de informação.

5 Considerações finais

Este trabalho abordou que o problema da crescente demanda por transportes nas cidades brasileiras pode ser tratado por um bom planejamento de transportes. Foi comentado sobre a importância dos modelos como ferramentas de apoio ao planejamento de transportes e sobre a importância de se considerar o sistema como um todo, observando as decisões individuais das pessoas e sua relação com o sistema. Nesse contexto, tratou-se da simulação como ferramenta de modelagem de transportes, que vem surgindo nos últimos anos, principalmente a simulação por agentes, que permite versatilidade e abrangência ao modelo.

A simulação de transportes baseada em agentes foi conceituada nos termos do MATSim, plataforma de simulação de transportes multi-agentes e de larga escala. Ferramenta esta que simula o trânsito através de agentes independentes, estes tem planos de atividades a executar ao longo do dia em diferentes localidades da cidade e encontram restrições ao longo da execução da sua rotina. A ferramenta, que requer um modelo de entrada baseado em uma rede e uma população inicial, busca o equilíbrio ao tentar otimizar os planos de todos os agentes simultaneamente.

A partir das especificações do MATSim e de dados iniciais acerca da rede, população e uso de solo, foi construído o modelo de Joinville. A construção deste foi a etapa principal do trabalho. Iniciou-se com a caracterização dos dados de entrada, daí partiu-se às etapas de tratamento e transcrição dos dados, onde foram encontrados muitos problemas que precisaram ser devidamente tratados.

O modelo construído foi finalmente simulado com o uso do MATSim. A simulação, que foi realizada com as configurações básicas do simulador, produziu como resultado os dados brutos denominados *eventos*. Estes dados puderam ser visualizados e analisados com o uso do Via, ferramenta externa que permite a visualização dos resultados do MATSim. Os dados foram agregados em alguns arcos e comparados a contagens de tráfego reais para avaliação do modelo e foi constatado empiricamente uma certa proximidade do modelo às contagens nos horários de pico. A análise dos dados agregados permitiu a constatação da versatilidade de informações que um modelo baseado em agentes pode entregar, evidenciando assim sua potencial aplicação no planejamento de transportes.

Pode-se dizer que a principal contribuição do trabalho está na aplicação de um simulador multi-agentes para modelar e simular o trânsito de uma cidade brasileira de porte médio a partir de dados de fontes e formatos diversos, que necessitaram de um processo de tratamento e transcrição.

5.1 Recomendações a estudos futuros

Como recomendações a trabalhos futuros, há primeiramente a calibração e atualização do modelo através da compreensão dos parâmetros de configuração do MATSim, do uso de novos dados e do uso da extensão de calibração *cadyts*. Há também a possibilidade de extensão do modelo, com a modelagem do transporte coletivo e dos semáforos, por exemplo. Há ainda a opção de explorar o valor dos eventos resultantes, a partir de diferentes análises agregadas, para o planejamento de transportes. Uma oportunidade também consiste na construção de uma interface que melhor permitiria o uso do modelo por usuários finais. Outra opção é explorar a possibilidade de uma rede variável no tempo, que poderia incluir incidentes de trânsito. Há também a melhor descrição da rede, com a inserção de lombadas e faixas de pedestre, e até mesmo uma melhor diferenciação das vias por pavimento e hierarquia. Por último, a interface entre este modelo e um segundo, para explorar a comunicação entre duas cidades ou até mesmo com um modelo feito com outro *software*.

Referências

AMERICAN ASSOCIATION OF STATE HIGHWAY AND TRANSPORTATION OFFICIALS (AASHTO). *A policy on geometric design of highways and streets.* Washington, D.C., 2004. Citado na página 35.

ASSOCIAÇÃO NACIONAL DE TRANSPORTES PÚBLICOS (ANTP). *Relatório Comparativo 2003-2012*. Vitória, 2014. Citado na página 23.

BALMER, M. *Travel Demand Modeling for Multi-agent Transport Simulations: Algorithms and Systems*. Tese (Doutorado) — Swiss Federal Institute of Technology of Zurich - ETHZ, Zürich, 2007. Citado 7 vezes nas páginas 23, 25, 26, 28, 31, 33 e 35.

BALMER, M.; AXHAUSEN, K.; NAGEL, K. An agent based demand modeling framework for large scale micro-simulations, paper submitted for the 85th annual meeting of the transportation research board. *TRB, Washington DC*, 2006. Citado na página 43.

BALMER, M.; BERNARD, M.; AXHAUSEN, K. Matching geo-coded graphs. In: *Swiss Transport Research Conference*. Ascona: Swiss Federal Institute of Technology, 2005. Citado na página 35.

BAZZAN, A. et al. Itsumo: an agent-based simulator for its applications. In: *Proc. of the 4th Workshop on Artificial Transportation Systems and Simulation*. [S.l.: s.n.], 2010. Citado na página 43.

DEPARTAMENTO ESTADUAL DE TRÂNSITO DE SANTA CATARINA. 2015. Disponível em: <http://www.detran.sc.gov.br/index.php/estatistica/veiculos>. Citado na página 41.

DOWLING, R. *Planning Techniques to Estimate Speeds and Service Volumes for Planning Applications*. [S.l.]: National Academy Press, 1997. (NCHRP report, Nº 387). United States Federal Highway Administration and American Association of State Highway and Transportation Officials and National Research Council (U.S.). Transportation Research Board and National Cooperative Highway Research Program. ISBN 9780309060585. Citado 2 vezes nas páginas 53 e 54.

ENGELBRECHT, A. *Computational Intelligence*: an introduction. [S.l.]: J. Wiley and Sons, 2002. Citado na página 34.

ENVIRONMENTAL SYSTEMS RESEARCH INSTITUTE (ESRI). *Shapefile*. [S.l.], 1998. Citado 2 vezes nas páginas 46 e 91.

FARINHA, P. M. L. *Modelos de Simulação em MATSim aplicados à análise de Sistemas de Transportes*. Dissertação (Mestrado) — Instituto Superior de Engenharia de Lisboa, Lisboa, 2013. Citado na página 23.

FERBER, J. *Multi-agent Systems: An Introduction to Distributed Artificial Intelligence*. [S.l.]: Addison-Wesley, 1999. ISBN 9780201360486. Citado na página 27.

FLYNN, P. *The XML FAQ*: Frequently-asked questions about the extensible markup language. [S.l.], 2006. Disponível em: <http://xml.silmaril.ie>. Citado na página 35.

FOURIE, P. J. *An initial implementation of a multi-agent transport simulator for South Africa*. Dissertação (Mestrado) — University of Pretoria, Pretoria, 2009. Citado na página 34.

FUNDAÇÃO INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATÍSTICA (IBGE). *Censo Demográfico*. [S.l.], 2010. Citado 4 vezes nas páginas 41, 45, 46 e 49.

FUNDAÇÃO INSTITUTO DE PESQUISA E PLANEJAMENTO PARA O DESENVOLVIMENTO SUSTENTÁVEL DE JOINVILLE (IPPUJ). *Plano de Mobilidade e Acessibilidade de Joinville*: diagnóstico. Joinville: Prefeitura Municipal, 2014. Citado 4 vezes nas páginas 41, 42, 43 e 51.

FUNDAÇÃO INSTITUTO DE PESQUISA E PLANEJAMENTO PARA O DESENVOLVIMENTO SUSTENTÁVEL DE JOINVILLE (IPPUJ); INSTITUTO DE PESQUISA CATARINENSE (IPC). *Pesquisa Origem-Destino Domiciliar 2010*. Joinville, 2010. Disponível em: <https://ippuj.joinville.sc.gov.br/arquivo/lista/codigo/8-Pesquisa%2BOrigem-Destino.html>. Citado 2 vezes nas páginas 41 e 46.

GOLDBARG, M. C. *Grafos*: conceitos, algoritmos e aplicações. Rio de Janeiro: Elsevier, 2012. ISBN 9788535257182. Citado na página 35.

GOOGLE. *Street View*. 2015. Disponível em: <https://www.google.com/maps/views/>. Citado na página 53.

HAGBERG, A. A.; SCHULT, D. A.; SWART, P. J. Exploring network structure, dynamics, and function using networkx. In: VAROQUAUX, G.; VAUGHT, T.; MILLMAN, J. (Ed.). *Proceedings of the 7th Python in Science Conference (SciPy2008)*. [S.l.: s.n.], 2008. p. 11–15. Citado na página 91.

HILLIER, F. S.; LIEBERMAN, G. J. *Introdução à pesquisa operacional*. 8. ed. São Paulo: McGraw Hill, 2006. Citado na página 34.

HOEL, L. A.; GARBER, N. J.; SADEK, A. W. *Engenharia de Infraestrutura de Transportes*: uma integração multimodal. São Paulo: Cengage Learning, 2011. ISBN 9788522110759. Citado 3 vezes nas páginas 35, 41 e 42.

IPPUJ. 2015. Disponível em: <https://ippuj.joinville.sc.gov.br>. Citado 9 vezes nas páginas 41, 45, 46, 47, 48, 49, 50, 52 e 53.

ITTNER, I. *Aplicação da Modelagem e Simulação Computacional como Pré-Experimento para Intervenções no Trânsito Urbano*. Joinville, 2014. Trabalho de Conclusão de Curso. Universidade do Estado de Santa Catarina. Citado na página 43.

JOINVILLE. *Lei Nº 1262, de 27 de abril de 1973*: reestrutura o plano diretor e dá outras providências. Joinville: Diário Oficial Eletrônico do Município de Joinville, v.126, n.66, p.6009, 8 abr., 1973. 44 p. Citado na página 41.

JOINVILLE. *Lei Complementar Nº 261, de 28 de fevereiro de 2008*: dispõe sobre as diretrizes estratégias e institui o plano diretor de desenvolvimento sustentável do município de joinville e dá outras providências. Joinville: Diário Oficial Eletrônico do Município de Joinville, v.126, n.66, p.6009, 8 abr., 2008. 67 p. Citado na página 41.

MANHEIM, M. *Fundamentals of Transportation Systems Analysis: Basic Concepts*. [S.l.]: MIT Press, 1979. (MIT Press Classic, v. 1). ISBN 9780262632898. Citado 2 vezes nas páginas 25 e 26.

MATSIM. 2015. Disponível em: <http://matsim.org/>. Citado 6 vezes nas páginas 23, 29, 40, 45, 52 e 91.

ORTÚZAR, J. de D.; WILLUMSEN, L. *Modelling Transport*. 4th. ed. [S.l.]: Wiley, 2011. ISBN 9781119993520. Citado 3 vezes nas páginas 23, 25 e 26.

PEREIRA, R. H. M.; SCHWANEN, T. Tempo de deslocamento casa-trabalho no brasil (1992-2009): diferenças entre regiões metropolitanas, níveis de renda e sexo. Instituto de Pesquisa Econômica Aplicada (IPEA), Brasília, n. 1813, 2013. Citado na página 23.

PREFEITURA MUNICIPAL DE JOINVILLE (PMJ). 2015. Disponível em: <https://www.joinville.sc.gov.br>. Citado 4 vezes nas páginas 45, 46, 50 e 91.

QGIS DEVELOPMENT TEAM. *QGIS Geographic Information System*. 2015. Open Source Geospatial Foundation Project. Disponível em: <http://qgis.osgeo.org>. Citado 2 vezes nas páginas 63 e 91.

RANEY, B. K. *Learning Framework for Large-scale Multi-agent Simulations*. Tese (Doutorado) — Swiss Federal Institute of Technology of Zurich - ETHZ, Zürich, 2005. Citado 4 vezes nas páginas 25, 27, 28 e 33.

RANEY, B. K.; NAGEL, K. An improved framework for large-scale multi-agent simulations of travel behaviour. In: RIETVELD, P.; JOURQUIN, B.; WESTIN, K. (Ed.). *Towards better performing European Transportation Systems*. London: Routledge, 2006. p. 305–347. Citado na página 29.

RIBEIRO, R. A. *Modelo baseado em agentes para estimar a geração e a distribuição de viagens intraurbanas*. Tese (Doutorado) — Universidade de São Paulo, 2011. Citado na página 43.

RIESER, M. *Adding Transit to an Agent-based Transportation Simulation: Concepts and Implementation*. Tese (Doutorado) — Technische Universität Berlin, Berlin, 2010. Citado 5 vezes nas páginas 28, 29, 30, 33 e 34.

RIESER, M. et al. *MATSim User Guide*. 0.6.1. ed. [S.l.], 2014. Citado 8 vezes nas páginas 29, 30, 32, 34, 35, 36, 38 e 39.

SENOZON AG. *Via*. 1.5. ed. Zürich, 2015. Disponível em: <http://via.senozon.com/>. Citado na página 68.

SILVA, G. R. L. *Sistema multiagente para simulação da dinâmica de estacionamentos-SMDES*. Dissertação (Mestrado) — Universidade Federal de Minas Gerais, Belo Horizonte, 2011. Citado na página 23.

SILVEIRA, F. K. *Controle Inteligente de Semáforos de Trânsito Utilizando Redes Neurais Artificiais com Funções de Base Radial*. Joinville, 2012. Trabalho de Conclusão de Curso. Universidade do Estado de Santa Catarina. Citado na página 43.

SYSTEMS, T.-T. S. *Aimsun 8 Dynamic Simulatiors Users' Manual*. [S.l.], 2014. Citado na página 43.

WEISS, G. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. [S.l.]: MIT Press, 1999. (Intelligent Robotics and Autonomous Agents Series). ISBN 9780262731317. Citado na página 27.

WORLD WIDE WEB CONSORTIUM. *eXtensible Markup Language (XML)*. [S.l.], 2006. Disponível em: <http://www.w3.org/XML/>. Citado 2 vezes nas páginas 35 e 91.

Anexos

ANEXO A – Conversão dos dados da rede viária

Iniciou-se pela conversão da rede ao formato especificado pelo MATSIM, no formato XML (W3C, 2006), porém com arcos bidirecionais ainda para testes. Foi adaptada a ferramenta com o código aberto e desenvolvida por (ref) para obter o resultado. Foi verificado que o grafo gerado era desconexo. Muitos arcos não estavam devidamente conectados no traçado da rede. Isso se deve ao fato de que os dados na PMJ (2015) são usados para fins de mapeamento e geoprocessamento, não para roteirização, então não existe uma exigência específica quanto ao grafo correspondente. Para contornar esse problema foi necessário a criação de uma ferramenta com o uso da biblioteca Hagberg, Schult e Swart (2008) que lesse o arquivo no formato Shapefile (ESRI, 1998), criasse o grafo e a seguir o dividisse em grafos desconexos e retornasse os mesmos para avaliação manual. Esse processo gerou mais de 300 grafos desconexos, sendo um deles a rede principal. Todos foram analisados manualmente usando o QGIS (2015). A maior parte dos subgrafos era irrelevante, becos-sem-saída que não afetariam o modelo e os que eram relevantes foram manualmente inseridos na rede principal. Completada esta etapa prosseguiu-se à inserção dos sentidos das vias, tornando o grafo direcionado. Mais uma vez o resultado foi um grafo desconexo. Desta vez a conexidade fraca era verificada devido ao processo anterior porém a conexidade forte, que determina que todos os nós devem ser alcançáveis a partir de qualquer nó da rede, não foi verificada. Para solucionar isso foi adaptada a ferramenta anterior e novamente foram gerados outros subgrafos, mais de 100. Um a um foram analisados e caso irrelevantes descartados ou caso contrário inseridos na rede principal. Por fim obteve-se o grafo fortemente conexo e com todos os atributos necessários ao MatSIM e dessa forma a rede foi convertida para simulação.

```
1  # Paths
2  pathNetwork = " " # Path to the network shapefile.
3  pathOutput = " " # Path to the output file.
4
5  G = nx.read_shp(pathNetwork)
6
7  D = nx.DiGraph()
8
9  # Fetches the meaningful data from G to add to D.
10 length = nx.get_edge_attributes(G, "Shape_Leng")
11 numLanes = nx.get_edge_attributes(G, "Faixa")
12 direction  = nx.get_edge_attributes(G, "ONEWAY")
13 Wkb = nx.get_edge_attributes(G, "Wkb")
14 obs = nx.get_edge_attributes(G, "OBS")
15 name = nx.get_edge_attributes(G, "NAME")
16 #neighbourhood = nx.get_edge_attributes(G, "Layer")
17
```

```
18 highways = ["BR. CENTO E UM", "EST DONA FRANCISCA", "JULIO STOLF",
19             "RODOVIA DO ARROZ", "RODOVIA SC 418", "RODOVIA SC-301"]
20
21 # Iterates through G to transform the edges to D accordingly to MATSim specifications (http://www.matsim.org/files/
       dtd/network_v1.dtd).
22 for e in G.edges():
23     startnode = e[0]
24     endnode = e[1]
25
26     # FT is "from-to". Carries edge direction information.
27     if direction[(startnode, endnode)] == 'TF':
28         if highways.count(name[(startnode, endnode)]):
29             cap = str(2000*(int(numLanes[(startnode, endnode)]))) # Capacity of the edges, calculated.
30             fspeed = "24.44" # Freespeed, determined.
31         elif int(numLanes[(startnode, endnode)]) > 1:
32             cap = str(750*(int(numLanes[(startnode, endnode)]))) # Capacity of the edges, calculated.
33             fspeed = "17.78" # Freespeed, determined.
34         else:
35             cap = str(550*(int(numLanes[(startnode, endnode)]))) # Capacity of the edges, calculated.
36             fspeed = "15.56" # Freespeed, determined.
37
38         D.add_edge(endnode, startnode, attr_dict=
39                    {'numLanes':numLanes[(startnode, endnode)],     # Number of edges.
40                     'direction': direction[(startnode, endnode)],  # Direction of edges.
41                     'name':name[(startnode, endnode)],             # Name of the street.
42                     'obs': obs[(startnode, endnode)],              # General observation.
43                     'Wkb':Wkb[(startnode, endnode)],               # Edges' shapes.
44                     'length': length[(startnode, endnode)],        # Edges' length.
45                     'capacity': cap,
46                     'freespeed': fspeed,
47                     'modes': "car"})                               # Modes, determined.
48
49     elif direction[(startnode, endnode)] == 'FT':
50         if highways.count(name[(startnode, endnode)]):
51             cap = str(2000*(int(numLanes[(startnode, endnode)]))) # Capacity of the edges, calculated.
52             fspeed = "24.44" # Freespeed, determined.
53         elif int(numLanes[(startnode, endnode)]) > 1:
54             cap = str(750*(int(numLanes[(startnode, endnode)]))) # Capacity of the edges, calculated.
55             fspeed = "17.78" # Freespeed, determined.
56         else:
57             cap = str(550*(int(numLanes[(startnode, endnode)]))) # Capacity of the edges, calculated.
58             fspeed = "15.56" # Freespeed, determined.
59
60         D.add_edge(startnode, endnode, attr_dict=
61                    {'numLanes':numLanes[(startnode, endnode)],
62                     'direction': direction[(startnode, endnode)],
63                     'name':name[(startnode, endnode)],
64                     'obs': obs[(startnode, endnode)],
65                     'Wkb':Wkb[(startnode, endnode)],
66                     'length': length[(startnode, endnode)],
67                     'capacity': cap,
68                     'freespeed': fspeed,
69                     'modes': "car"})
70
71     else:
72         if obs[(startnode, endnode)] == '2 TF 1 FT': # For a special kind of edges where 2 lanes are in a direction
               and 1 in the opposite.
73             D.add_edge(startnode, endnode, attr_dict=
74                    {'numLanes':'1',
75                     'direction': direction[(startnode, endnode)],
```

```
                    'name':name[(startnode, endnode)],
                    'obs': obs[(startnode, endnode)],
                    'Wkb':Wkb[(startnode, endnode)],
                    'length': length[(startnode, endnode)],
                    'capacity': "550",
                    'freespeed': "15.56",
                    'modes': "car"})

            D.add_edge(endnode, startnode, attr_dict=
                    {'numLanes':'2',
                    'direction': direction[(startnode, endnode)],
                    'name':name[(startnode, endnode)],
                    'obs': obs[(startnode, endnode)],
                    'Wkb':Wkb[(startnode, endnode)],
                    'length': length[(startnode, endnode)],
                    'capacity': "1400",
                    'freespeed': "17.78",
                    'modes': "car"})

        else: # Undirected edges.
            if highways.count(name[(startnode, endnode)]):
                cap = str(750*(int(numLanes[(startnode, endnode)]))) # Capacity of the edges, calculated.
                fspeed = "22.22" # Freespeed, determined.
            elif int(numLanes[(startnode, endnode)]) > 2:
                cap = str(700*0.5*(int(numLanes[(startnode, endnode)]))) # Capacity of the edges, calculated.
                fspeed = "17.78" # Freespeed, determined.
            else:
                cap = str(550*0.5*(int(numLanes[(startnode, endnode)]))) # Capacity of the edges, calculated.
                fspeed = "15.56" # Freespeed, determined.

            D.add_edge(startnode, endnode, attr_dict=
                    {'numLanes':str(0.5*(int(numLanes[(startnode, endnode)]))),
                    'direction': direction[(startnode, endnode)],
                    'name':name[(startnode, endnode)],
                    'obs': obs[(startnode, endnode)],
                    'Wkb':Wkb[(startnode, endnode)],
                    'length': length[(startnode, endnode)],
                    'capacity': cap,
                    'freespeed': fspeed,
                    'modes': "car"})

            D.add_edge(endnode, startnode, attr_dict=
                    {'numLanes':str(0.5*(int(numLanes[(startnode, endnode)]))),
                    'direction': direction[(startnode, endnode)],
                    'name':name[(startnode, endnode)],
                    'obs': obs[(startnode, endnode)],
                    'Wkb':Wkb[(startnode, endnode)],
                    'length': length[(startnode, endnode)],
                    'capacity': cap,
                    'freespeed': fspeed,
                    'modes': "car"})

# Bursts the graph into it's strongly connected components.
D_subgraphs = list(nx.strongly_connected_component_subgraphs(D, True))

for S in D_subgraphs:
    # We know from previous information that only the main subgraph is enough for the network so we discard the rest.
    if len(S.edges()) > 1000:

        S = nx.convert_node_labels_to_integers(S, label_attribute = "coord")
```

```
136
137         # Creates the Element Tree object for pretty-print output of the network output file.
138         network = ET.Element("network",
139                 attrib={'name':"Joinville/SC."})
140
141         # Nodes element to store nodes info.
142         nodes = ET.SubElement(network, "nodes")
143
144         for i in range(len(S)):
145             ET.SubElement(nodes, "node",
146                 attrib={'id': str(i),
147                         'x': str(S.node[i]['coord'][0]),
148                         'y': str(S.node[i]['coord'][1]),
149                         'type':"2"})
150
151         # Links element to store links info.
152         links = ET.SubElement(network, "links",
153                 attrib={'capperiod': "01:00:00",
154                         'effectivecellsize': "7.5",
155                         'effectivelanewidth': "3.75"})
156
157         # Fetches the meaningful data to output in the MATSim network.
158         length = nx.get_edge_attributes(S, "length")
159         numLanes = nx.get_edge_attributes(S, "numLanes")
160         capacity = nx.get_edge_attributes(S, "capacity")
161         freespeed = nx.get_edge_attributes(S, "freespeed")
162
163         # Iterates over the edges of the graph to gather it's information and create the links in the Network Element
                Tree.
164         j = 1
165         for e in S.edges():
166             startnode = e[0]
167             endnode = e[1]
168
169             # Since the directions were already fixed before, here we simply need the data.
170             ET.SubElement(links, "link", attrib={
171                         'id': str(j),
172                         'from': str(startnode),
173                         'to':   str(endnode),
174                         'capacity': str(capacity[(startnode, endnode)]),
175                         'permlanes': str(numLanes[(startnode, endnode)]),
176                         'modes': "car",
177                         'oneway': str(1),
178                         'type': str(10),
179                         'length': str(length[(startnode, endnode)]),
180                         'freespeed': str(freespeed[(startnode, endnode)])})
181             j = j + 1
182
183             tree = ET.ElementTree(network)
184
185         # Outputs the network to the XML file.
186         with open(pathOutput, 'wb') as f:
187             st = """<?xml version="1.0" encoding="UTF-8" ?>
188                     <!DOCTYPE network SYSTEM "http://www.matsim.org/files/dtd/network_v1.dtd">
189                     """
190             f.write(st.encode(encoding='utf_8', errors='strict'))
```

ANEXO B – Transcrição dos dados da OD

```
1  class Population():
2      def ___init___(self, fileOD_, fileNeighbourhoods_, ouputPath_):
3
4          # Strings that hold the input files . Origin-destination matrix file and neighbourhoods information file.
5          self.fileOD = fileOD_
6          self.fileNeighbourhoods = fileNeighbourhoods_
7          self.ouputPath = ouputPath_
8      # A list to store the created Neighbourhood() objects.
9      neighbourhoodsList = []
10     # A list to store the created agents Ids. It is used to avoid creation of repeated agents in the method
           printAgent().
11     idList = []
12     peopleProcessed = 0
13     # Number of agents with or without car and or employment.
14
15     # First method. Will call the others.
16     def addPopulation(self):
17         # This object be used to split the fileOD by neighbourhood.
18         matrixOD = ""
19
20         # This block will create a Neighbourhood() object for each neighbourhood contained in the neighbourhoodsFile.
21         for line in open(self.fileNeighbourhoods):
22             lineSplit = line.split('\t')
23             name = lineSplit[1]
24             idNeighbourhood = int(lineSplit[0])
25             population = int(lineSplit[2])
26             familiesList_ = []
27
28             neighbourhood = Neighbourhood(idNeighbourhood, name, population, familiesList_)
29             self.neighbourhoodsList.append(neighbourhood)
30
31         # Will extract a origin-destination matrix for each Neighbourhood() object and then call the
               addNeighbourhoodPopulation method.
32         for neighbourhood in self.neighbourhoodsList:
33             for line in open(self.fileOD, 'rb'):
34                 # Used to decode the fileOD from it's enconding to unicode.
35                 lineDecode = line.decode('latin-1')
36                 lineDecodeSplit = lineDecode.split('\r')
37                 for line2 in lineDecodeSplit:
38                     lineSplit2 = line2.split('\t')
39                     if lineSplit2[1] == neighbourhood.name:
40                         matrixOD = matrixOD + line2 + '\n'
41
42             self.addNeighbourhoodPopulation(neighbourhood, matrixOD)
43             print("neighbourhood: ", neighbourhood.name)
44             matrixOD = ""
45
46
47
48     # This method will create the population of each neighbourhood.
49     def addNeighbourhoodPopulation(self, neighbourhood, matrixOD):
50         # Splits the neighbourhood OD into lines then pops the last line because it is empty (caused by the previous
               method).
51         matrixODSplit = matrixOD.split('\n')
```

```
52          matrixODSplit.pop()
53          # Saves the family Ids of the neighbourhood.
54          familyIdList = []
55          # This ibject is used to split the matrixOD by family.
56          tableOD = ""
57          cnt = 0
58
59          # Iterates among the matrixOD and saves the unique family IDs.
60          for line in matrixODSplit:
61              lineSplit = line.split('\t')
62              if familyIdList.count(int(lineSplit[2])) == 0:
63                  familyIdList.append(int(lineSplit[2]))
64          neighbourhood.numFamilies = len(familyIdList)
65
66          # Iterates among family IDs and calls the addFamily method.
67          for familyId in familyIdList:
68              for line in matrixODSplit:
69                  lineSplit = line.split('\t')
70                  if int(lineSplit[2]) == familyId:
71                      tableOD = tableOD + line + '\n'
72              neighbourhood.addFamily(tableOD)
73              print("family n ", cnt)
74              cnt += 1
75              tableOD = ""
76
77          print("total people in ", neighbourhood.name, ": ", neighbourhood.peopleProcessed)
78
79  # This class is for the Neighbourhood() object that stores the neighbourhoods information.
80  class Neighbourhood(Population):
81      def __init__(self, idNeighbourhood_, name_, population_, familiesList_):
82          self.idNeighbourhood = idNeighbourhood_
83          self.name = name_
84          self.population = population_
85          self.familiesList = familiesList_
86
87      # Adds the families agents.
88      def addFamily(self, tableOD):
89          # idF is the family id and idFamily is it's neighbourhood's id concatenated with it's own id.
90          idF = tableOD.split('\t')[2]
91          idFamily = str(self.idNeighbourhood) + idF
92          if idFamily == "1121264":
93              pass
94          # Creates the object Family that will store the agents.
95          family = Family(idFamily, self.name)
96          tableODMatrix = []
97          previousPerson = 0
98          routineList = []
99          cnt = 0
100
101         # Transforms the tableOD into a matrix.
102         tableODLineSplit = tableOD.split('\n')
103         for line in tableODLineSplit:
104             lineSplit = line.split('\t')
105             tableODMatrix.append(lineSplit)
106
107         # Iterates through the tableOD and calls addAgent for each unique person.
108         for lineOD in tableODMatrix:
109             # idP_fP is a tuple containing the persons id and it's family position. It's used to look for unique
                    persons in the tableOD.
110             try: idP_fP = [lineOD[3], lineOD[4]]
```

```
111                 # When reaches the tableOD's end.
112                 except: idP_fP = [0,0]
113
114                 # Fires when the iterator reaches a new person and adds the previous person as a new agent.
115                 if idP_fP != previousPerson and previousPerson != 0:
116                     cnt += 1
117                     family.addAgent(previousPerson[0], previousPerson[1], routineList, self.idNeighbourhood)
118                     self.peopleProcessed += 1
119                     print("agent n ", cnt)
120                     if idP_fP == [0,0]: break
121                     routineList = []
122                 # Adds each instance of the persons's routine to a list.
123                 routineList.append([lineOD[5], lineOD[6], lineOD[7], lineOD[8], lineOD[9], lineOD[10], lineOD[11]])
124                 previousPerson = idP_fP
125
126
127     # Used in the addAgent method.
128     def findNeighbourhood(self, name_):
129         for it in self.neighbourhoodsList:
130             if name_ == it.name:
131                 return True
132         return False
133
134 # Class with the Family() object.
135 class Family(Neighbourhood):
136     def ___init___(self, idFamily_, neighbourhood_):
137         self.idFamily = idFamily_
138         self.neighbourhood = neighbourhood_
139
140     # Adds agents to a family.
141     # This method precisely checks the input routine for different kinds of errors.
142     # In case there are no errors it creates the agent, otherwise it calls the exceptionHandler method.
143     def addAgent(self, idPerson, familyPosition, routineList, idNeighb):
144         # Counts the amount of origin or destination neighbourhoods that are not actually neghbourhoods (not in
                neighbourhoodsList).
145         neighbourhoodException = 0
146         # Next two exceptions count how many arrival or departure times are out of the expected format.
147         departureTimeFormatException = 0
148         arrivalTimeFormatException = 0
149         # True if first departure is not from home.
150         firstActivityDepartureException = False
151         # True if last arrival is not to home.
152         lastActivityArrivalException = False
153         # True if the agent has any departures before previous arrival.
154         activityTimeOrderException = False
155         # Counts the number of departures that run after it's respective arrival.
156         activityTimeBoundsException = 0
157         # Counts the number of departures that run before the previous activity's respective arrival.
158         activityTimeSequenceException = 0
159         # True if first activity's departure is before 3 AM. Doesn't behave well in the model.
160         activityDepartureTimeLimitException = False
161         # True if last activity's arrival is before 6 AM. Doesn't behave well in the model.
162         activityArrivalTimeLimitException = False
163         # Counts the number of non matching activities in the routine time sequence. E.g.: home-work then leisure-
                home -> missing one leg.
164         activityChainException = 0
165         # True if any of the above is True or more than 0.
166         exception = False
167         # True if any of the agent's legs are by car.
168         foundCar = False
```

```
169         # True if any of the agent's activity is Work.
170         worker = False
171
172         # Iterates through the routineList and looks for a few exceptions.
173         for it in routineList:
174             if not self.findNeighbourhood(it[0]):
175                 neighbourhoodException += 1
176                 exception = True
177             if not self.findNeighbourhood(it[1]):
178                 neighbourhoodException += 1
179                 exception = True
180
181             # Besides checking for the departureTimeFormatException also converts the time format from hh_mm_ss
                    to the integer amount of seconds.
182             if not isinstance(it[4], int):
183                 try:
184                     it[4] = list(map(int, it[4].split("_")))
185                     it[4] = 3600*it[4][0] + 60*it[4][1] + it[4][2]
186                 except:
187                     departureTimeFormatException += 1
188                     exception = True
189
190             # Similarly as the above block but for arrival times.
191             if not isinstance(it[5], int):
192                 try:
193                     it[5] = list(map(int, it[5].split("_")))
194                     it[5] = 3600*it[5][0] + 60*it[5][1] + it[5][2]
195                 except:
196                     arrivalTimeFormatException += 1
197                     exception = True
198
199             if isinstance(it[4], int):
200                 if int(it[4]) <= 10800:
201                     activityDepartureTimeLimitException = exception = True
202
203             if isinstance(it[5], int):
204                 if int(it[5]) <= 18000 and it[3] == "Residencia":
205                     activityArrivalTimeLimitException = exception = True
206
207             try:
208                 if int(it[4]) >= int(it[5]):
209                     activityTimeBoundsException += 1
210                     exception = True
211             except:
212                 pass
213
214         # Will check for the next exceptions only if time formats are correct.
215         if not (departureTimeFormatException or arrivalTimeFormatException):
216             # Sorts the routineList by departure times.
217             routineList.sort(key=operator.itemgetter(4))
218             routineListAux = routineList.copy()
219             routineListAux.sort(key=operator.itemgetter(5))
220
221             # Compares the routineList with a similar one sorted by arrival times.
222             if routineList != routineListAux:
223                 activityTimeOrderException = exception = True
224
225             if not routineList[0][2] == "Residencia":
226                 firstActivityDepartureException = exception = True
227
```

```
228             if not routineList[-1][3] == "Residencia":
229                 lastActivityArrivalException = exception = True
230
231             for i in range(1, len(routineList)):
232                 if routineList[i][2] != routineList[i-1][3]:
233                     activityChainException += 1
234                     exception = True
235                 if isinstance(it[4], int) and isinstance(it[5], int):
236                     if routineList[i][4] <= routineList[i-1][5]:
237                         activityTimeSequenceException += 1
238                         exception = True
239
240         # Checks if there were any exceptions.
241         # If so, saves the agent's exceptions to a list and calls the exceptionHandler.
242         if exception == True:
243             exceptionList = [neighbourhoodException, departureTimeFormatException, arrivalTimeFormatException,
244                              firstActivityDepartureException, lastActivityArrivalException,
                                     activityTimeOrderException,
245                              activityChainException, activityTimeBoundsException,
                                     activityDepartureTimeLimitException,
246                              activityArrivalTimeLimitException, activityTimeSequenceException]
247
248             self.exceptionHandler(exceptionList, idPerson, familyPosition, routineList, idNeighb)
249
250         # Checks if the agent has a car leg and in case yes adds him to the family. Else count's him for reporting.
251         else:
252             for item in routineList:
253                 if item[6] == "Automovel" or item[6] == "Caminhao" or item[6] == "Taxi":
254                     foundCar = True
255                 if item[3] == "Trabalho":
256                     worker = True
257             if foundCar == True:
258                 if worker == True:
259                     agent = Agent(idPerson, familyPosition, routineList)
260                     agent.printAgent(self.idFamily)
261                     print("carWorkerAgent")
262                 else:
263                     agent = Agent(idPerson, familyPosition, routineList)
264                     agent.printAgent(self.idFamily)
265                     print("carNonWorkerAgent")
266             else:
267                 if worker == True:
268                     print("nonCarWorkerAgent")
269                 else:
270                     print("nonCarNonWorkerAgent")
271
272
273     # This method will handle the exceptions.
274     # For a few types it will correct it and call the addAgent again, for others will only report.
275     def exceptionHandler(self, exceptionList, idPerson, familyPosition, routineList, idNeighb):
276
277         # activityDepartureTimeLimitException
278         if exceptionList[8] > 0 and exceptionList[1] == 0 and exceptionList[2] == 0:
279             for it in routineList:
280                 # Checks if it has any departure before 3 AM, if yes sets it to 3h01AM.
281                 if it[4] <= 10800: # saida de casa antes das 3 da manha
282                     dif = it[5] - it[4]
283                     it[4] = 10860
284                     it[5] = it[4] + dif
285             try:
```

```
286                 return self.addAgent(idPerson, familyPosition, routineList, idNeighb)
287             except:
288                 pass
289
290         # activityArrivalTimeLimitException
291         if (exceptionList[9] > 0 and exceptionList[1] == 0 and exceptionList[2] == 0):
292             for it in routineList:
293                 # Checks if it has any arrival before 5 AM, if yes adds 24h to it to indicate that it is in the next
                        day.
294                 if it[5] <= 18000: # retorno para casa antes das 5 da manha
295                     it[5] += 86400
296                     it[4] += 86400
297             try:
298                 return self.addAgent(idPerson, familyPosition, routineList, idNeighb)
299             except:
300                 pass
301
302         # firstActivityDepartureException
303         # Adds the home activity before the first activity.
304         if exceptionList[3] == True and exceptionList[7] == 0:
305             destination = routineList[0][0]
306             mode = routineList[0][6]
307             activityEnd = routineList[0][4]
308             activity = routineList[0][2]
309
310             home = self.neighbourhood
311
312             if activity == "Trabalho":
313                 dur = 9*3600
314             elif activity == "Escola/curso":
315                 dur = 4*3600
316             elif (activity == "Compras" or
317                   activity == "Assuntos pessoais" or
318                   activity == "Saude"):
319                 dur = 2*3600
320             else: # "Visitar amigos" e "Lazer/passeio"
321                 dur = 3*3600
322
323             arrival = activityEnd - dur
324             departure = arrival - 1800
325
326             trip = [home, destination, "Residencia", activity, departure, arrival, mode]
327             routineList.insert(0, trip)
328             return self.addAgent(idPerson, familyPosition, routineList, idNeighb)
329
330         # lastActivityArrivalException
331         # Adds the home activity after the last activity.
332         if exceptionList[4] == True and exceptionList[7] == 0:
333             origin = routineList[-1][1]
334             mode = routineList[-1][6]
335             activityStart = routineList[-1][5]
336             activity = routineList[-1][3]
337
338             for it in routineList:
339                 if it[2] == "Residencia":
340                     home = it[0]
341                     break
342                 elif it[3] == "Residencia":
343                     home = it[1]
344                     break
```

```
345
346                 if activity == "Trabalho":
347                     dur = 9*3600
348                 elif activity == "Escola/curso":
349                     dur = 4*3600
350                 elif (activity == "Compras" or
351                       activity == "Assuntos pessoais" or
352                       activity == "Saude"):
353                     dur = 2*3600
354                 else: # "Visitar amigos" e "Lazer/passeio"
355                     dur = 3*3600
356
357                 departure = int(activityStart) + dur
358                 arrival = departure + 1800
359
360                 trip = [origin, home, activity, "Residencia", departure, arrival, mode]
361                 routineList.append(trip)
362                 return self.addAgent(idPerson, familyPosition, routineList, idNeighb)
363
364             exceptions = ("neighbourhoodException",                 "departureTimeFormatException",
365                           "arrivalTimeFormatException",             "firstActivityDepartureException",
366                           "lastActivityArrivalException",           "activityTimeOrderException",
367                           "activityChainException",                 "activityTimeBoundsException",
368                           "activityDepartureTimeLimitException",    "activityArrivalTimeLimitException",
369                           "activityTimeSequenceException")
370
371             # Reports the remaining exceptions.
372             print(self.idFamily)
373             for i in range(len(exceptionList)):
374                 if exceptionList[i] > 0:
375                     print(exceptions[i], exceptionList[i])
376
377 # Class with the Agent() object.
378 class Agent(Family):
379     def __init__(self, idAgent_, familyPosition_, routineList_):
380         self.idAgent = idAgent_
381         self.familyPosition = familyPosition_
382         self.routineList = routineList_
383
384     # Prints the agent to the Origin-Destination output file.
385     def printAgent(self, idFamily):
386         # Indexes for family position.
387         if self.familyPosition == "Agregado":
388             idPos = "01"
389         elif self.familyPosition == "Amigo(a)":
390             idPos = "02"
391         elif self.familyPosition == "Avo (o)":
392             idPos = "03"
393         elif self.familyPosition == "Chefe de familia":
394             idPos = "04"
395         elif self.familyPosition == "Conjuge":
396             idPos = "05"
397         elif self.familyPosition == "Empregado":
398             idPos = "06"
399         elif self.familyPosition == "Enteado":
400             idPos = "07"
401         elif self.familyPosition == "Filho(a)":
402             idPos = "08"
403         elif self.familyPosition == "Genro/Nora":
404             idPos = "09"
```

```
405             elif self.familyPosition == "Irmao(a)":
406                 idPos = "10"
407             elif self.familyPosition == "Mae/pai":
408                 idPos = "11"
409             elif self.familyPosition == "Neto(a)":
410                 idPos = "12"
411             elif self.familyPosition == "Parente":
412                 idPos = "13"
413             elif self.familyPosition == "Pensionista":
414                 idPos = "14"
415             elif self.familyPosition == "Sobrinho(a)":
416                 idPos = "15"
417             elif self.familyPosition == "Sogro(a)":
418                 idPos = "16"
419
420             idA = idFamily + self.idAgent + idPos
421
422             # Makes sure an agent is not added twice.
423             if not self.idList.count(idA):
424                 self.idList.append(idA)
425
426                 with open(self.pathOutput, 'ab') as f:
427
428                     for i in range(len(self.routineList)-1):
429                         activityDuration = str(float(self.routineList[i+1][4]) - float(self.routineList[i][5]))
430
431                         if self.routineList[i][3] == "Residencia":
432                             activityType = 'home'
433                         elif self.routineList[i][3] == "Trabalho":
434                             activityType = 'work'
435                         elif self.routineList[i][3] == "Escola/curso":
436                             activityType = 'education'
437                         elif (self.routineList[i][3] == "Compras" or
438                               self.routineList[i][3] == "Assuntos pessoais" or
439                               self.routineList[i][3] == "Saude"):
440                             activityType = 'shop'
441                         else:
442                             activityType = 'leisure'
443
444                         mode = "car"
445                         # First activity in the routine (home).
446                         if i == 0:
447                             activityHome1 = idA + "\t" + "home" + "\t" + str(self.routineList[i][4]) + "\t" + "-1" + "\n"
448                             activity = idA + "\t" + activityType + "\t" + activityDuration + "\t" + mode + "\n"
449                             f.write(activityHome1.encode(encoding='utf_8', errors='strict'))
450                             f.write(activity.encode(encoding='utf_8', errors='strict'))
451                             # For the case where the routine has only two activities.
452                             if i == (len(self.routineList)-2):
453                                 activityHome2 = idA + "\t" + "home" + "\t" + "-1" + "\t" + mode + "\n"
454                                 f.write(activityHome2.encode(encoding='utf_8', errors='strict'))
455                         # Last activity in the routine (home).
456                         elif i == (len(self.routineList)-2):
457                             activityHome2 = idA + "\t" + "home" + "\t" + "-1" + "\t" + mode + "\n" # same mode as
                                    last activity
458                             activity = idA + "\t" + activityType + "\t" + activityDuration + "\t" + mode + "\n"
459                             f.write(activity.encode(encoding='utf_8', errors='strict'))
460                             f.write(activityHome2.encode(encoding='utf_8', errors='strict'))
461                         # Other activities.
462                         else:
463                             activity = idA + "\t" + activityType + "\t" + activityDuration + "\t" + mode + "\n"
```

```
464                             f.write(activity.encode(encoding='utf_8', errors='strict'))
465
466
467 ### End of Classes. Execution starts from here ###
468
469 # Paths
470 pathOD = " " # Path to the Origin-Destination file.
471 pathNeighbourhood = " " # Path to the neighbourhood information file.
472 pathOutput = " " # Path to the output file.
473
474 pop = Population(pathOD, pathNeighbourhood, pathOutput)
475 pop.addPopulation()
```

ANEXO C – Conversão dos dados de uso de solo

```
# This scripts converts the land use data do a format treatable by the MATSim CreateFacilities class.
# The CreateFacilities class is available at MATSim's tutorial at www.matsim.org.

# Input files.
pathLandUse = " " # Path to the Land Use file (Sistema de Gestao Cadastral).
pathNeighbourhoods = " " # Path to the neighbourhood information file.
# Output location.
pathOutput = " " # Path to the output file.

f = open(pathLandUse)
n = open(pathNeighbourhoods)
o = open(pathOutput, "a")

# Writes header.
o.write('id\tCoordX\tCoordY\tTypes\n')
cnt = 0

# Skips both headers
f.readline()
n.readline()

nlines = n.readlines()

# Goes through all land units, reads the info, cleans and transforms, then outputs.
for line in f:
    lineSplit = line.split(";")
    neighbourhood = lineSplit[-3]
    # For finding the neighbourhood ID.
    for i in nlines:
        nlinesSplit = i.split("\t")
        if neighbourhood == nlinesSplit[1]:
            idF = nlinesSplit[0]

    # Checks all the land use options and saves into a list.
    types = []
    types.clear()
    if int(lineSplit[6]):
        if types.count('home') == 0:
            types.append('home')
    if int(lineSplit[7]):
        if types.count('work') == 0:
            types.append('work')
        if types.count('shop') == 0:
            types.append('shop')
    if int(lineSplit[8]):
        if types.count('work') == 0:
            types.append('work')
    if int(lineSplit[9]):
        if types.count('work') == 0:
            types.append('work')
        if types.count('shop') == 0:
```

```
52                  types.append('shop')
53          if int(lineSplit[10]):
54              if types.count('work') == 0:
55                  types.append('work')
56              if types.count('shop') == 0:
57                  types.append('shop')
58          if int(lineSplit[11]):
59              if types.count('work') == 0:
60                  types.append('work')
61              if types.count('education') == 0:
62                  types.append('education')
63          if int(lineSplit[12]):
64              if types.count('work') == 0:
65                  types.append('work')
66              if types.count(' leisure ') == 0:
67                  types.append('leisure ')
68          if int(lineSplit[13]):
69              if types.count(' leisure ') == 0:
70                  types.append('leisure ')
71          if int(lineSplit[14]):
72              if types.count('work') == 0:
73                  types.append('work')
74              if types.count('shop') == 0:
75                  types.append('shop')
76          if int(lineSplit[15]):
77              if types.count('work') == 0:
78                  types.append('work')
79              if types.count(' leisure ') == 0:
80                  types.append('leisure ')
81          if int(lineSplit[16]):
82              if types.count('work') == 0:
83                  types.append('work')
84
85          # Checks if the land unit has any use at all then writes it to the output.
86          if len(types) > 0:
87              o.write(str(idF) + str(cnt) + '\t' + lineSplit[21] + '\t' + lineSplit[22] + '\t' + ','.join(types) + '\n')
88          cnt = cnt + 1
89
90  print("Finished!")
```

ANEXO D – Conversão dos dados do censo demográfico

```
1  # Paths
2  pathOD = " " # Path to the Origin-Destination file.
3  pathCensus = " " # Path to the census file.
4  pathNeighbourhood = " " # Path to the neighbourhood information file.
5  pathOutput = " " # Path to the output file.
6
7
8  # Input files.
9  OD = open(pathOD)
10 CN = open(pathCensus)
11 N = open(pathNeighbourhoods)
12 # Output location.
13 O = open(pathOutput, "a")
14
15 CN.readline() # skip header
16 CNLines = CN.readlines()
17
18 # Neighbourhood name.
19 previousName = 0
20 """
21 The following variable stores the name of the neighbourhood, it's population and the cumulative proportion of persons
       per age group.
22 nTotals[0] -> Neighbourhood name
23 nTotals[1] -> Total population
24 nTotals[2] -> 0 to 5 years
25 nTotals[3] -> 6 to 9 years
26 nTotals[4] -> 10 to 14 years
27 nTotals[5] -> 15 to 17 years
28 nTotals[6] -> 18 to 24 years
29 nTotals[7] -> 25 to 34 years
30 nTotals[8] -> 35 to 44 years
31 nTotals[9] -> 45 to 59 years
32 nTotals[10] -> 60 to 64 years
33 nTotals[11] -> over 64 years old
34 """
35 nTotals = [0]*12
36 # A matrix that will store the nTotals of each neighbourhood.
37 nProportions = []
38
39 # Each line in CNLines corresponds to a censitary zone.
40 for line in CNLines:
41     lineS = line.split(";")
42     name = lineS[1] # neighbourhood name.
43
44     """
45     After going through all lines,
46     calculates the total population,
47     calculates the cumulative proportion of each age group and saves nTotals to nPropostions."""
48     if name != previousName and previousName != 0:
49         # Sums all age groups for the total.
50         for i in range(2, len(nTotals)):
```

```
51              nTotals[1] += nTotals[i]
52          # Calculates the simple proportion of each age group.
53          for i in range(2, len(nTotals)):
54              nTotals[i] = nTotals[i]/nTotals[1]
55          # Calculates the cumulative proportion by adding up to 1 (100% of the population).
56          for i in range(3, len(nTotals)):
57              nTotals[i] += nTotals[i-1]
58          nTotals[0] = previousName
59          nProportions.append(nTotals)
60          nTotals = [0]*12
61
62      # Goes through the census file reading the age count values and adds up to nTotals.
63      for i in range(24,137):
64          if i in range(24,42):
65              nTotals[2] += int(lineS[i])
66          if i in range(42,46):
67              nTotals[3] += int(lineS[i])
68          if i in range(46,51):
69              nTotals[4] += int(lineS[i])
70          if i in range(51,54):
71              nTotals[5] += int(lineS[i])
72          if i in range(54,61):
73              nTotals[6] += int(lineS[i])
74          if i in range(61,71):
75              nTotals[7] += int(lineS[i])
76          if i in range(71,81):
77              nTotals[8] += int(lineS[i])
78          if i in range(81,96):
79              nTotals[9] += int(lineS[i])
80          if i in range(96,101):
81              nTotals[10] += int(lineS[i])
82          if i in range(101,137):
83              nTotals[11] += int(lineS[i])
84
85      previousName = name
86
87      # Since the logic above leaves the last neighbourhood behind I had to create this extra block ahead.
88      if name == "zona_industrial_tupy":
89          for i in range(2, len(nTotals)):
90              nTotals[0] = nTotals[0] + nTotals[i]
91          for i in range(2, len(nTotals)):
92              nTotals[i] = nTotals[i]/nTotals[0]
93          for i in range(3, len(nTotals)):
94              nTotals[i] = nTotals[i] + nTotals[i-1]
95          nTotals[0] = previousName
96          nProportions.append(nTotals)
97
98  # OD proportions.
99  OD.readline() # skip header.
100 ODLines = OD.readlines()
101
102 #The following block will do the same as above but for work neighbourhood cumulative proportions.
103 previousNeighbourhood = 0 # For the name.
104 previousPersonId = 0
105 # Same as nTotals. Position 0 is for the name, 1 is for the total.
106 # This variable stores the cumulative proportion of people from each neighbourhood that works in each neighbourhood
        of the city.
107 wTotals = [0]*44
108 # Same as nProportions.
109 wProportions = []
```

```
110 # Used to differentiate workers from non-workers.
111 worker = False
112 # Used to save worker's work neighbourhood location.
113 workNeighbourhood = 0
114 # All 42 neighbourhoods of Joinville.
115 nNames = ["adhemar_garcia","america","anita_garibaldi","atiradores","aventureiro",
116             "boa_vista","boehmerwald","bom_retiro","bucarein","centro",
117             "comasa","costa_e_silva","espinheiros","fatima","floresta ",
118             "gloria","guanabara","iririu ","itaum","itinga ",
119             "jardim_iririu","jardim_paraiso","jardim_sofia","jarivatuba","joao_costa",
120             "morro_do_meio","nova_brasilia","paranaguamirim","parque_guarani","petropolis",
121             "pirabeiraba","profipo","rio_bonito","saguacu","santa_catarina",
122             "santo_antonio","sao_marcos","ulysses_guimaraes","vila_cubatao","vila_nova",
123             "zona_industrial_norte","zona_industrial_tupy"]
124
125 # Each line in ODLines represents a person's trip.
126 for line in ODLines:
127     lineS = line.split("\t")
128     neighbourhood = lineS[1] # Neighbourhood's name.
129     personId = lineS[4] # Person's unique ID.
130
131     # When moving to the next neighbourhood, sums up the total workers in the neighbourhood and calculates it's
            cumulative proportions.
132     if neighbourhood != previousNeighbourhood and previousNeighbourhood != 0:
133         # Sums to check the total number of workers in all neighbourhoods.
134         for i in range(2, len(wTotals)):
135             wTotals[1] += wTotals[i]
136         # Calculates the simple worker population proportions to each neighbourhood.
137         for i in range(2, len(wTotals)):
138             wTotals[i] = wTotals[i]/wTotals[1]
139         # Calculates the cumulative proportion.
140         for i in range(3, len(wTotals)):
141             wTotals[i] = wTotals[i] + wTotals[i-1]
142         wTotals[0] = previousNeighbourhood
143         wProportions.append(wTotals)
144         wTotals = [0]*44
145
146     # When moving to the next person, if it was a worker, counts it's neighbourhood up.
147     if personId != previousPersonId and previousPersonId != 0 and worker:
148         wTotals[(nNames.index(workNeighbourhood))+2] += 1
149         worker = False
150
151     # Checks if the person is a worker and saves his work location.
152     if lineS[9] == "Trabalho":
153         worker = True
154         workNeighbourhood = lineS[7]
155
156     previousPersonId = personId
157     previousNeighbourhood = neighbourhood
158
159 # This block will calculate the cumulative work proportions for the entire city.
160 cityWorkProportions = ["Joinville"] + [0]*43
161
162 # Calculates the total people that work in each neighbourhood.
163 for item in wProportions:
164     for i in range(3, len(item)):
165         cityWorkProportions[i] += (item[i]-item[i-1])*item[1]
166     cityWorkProportions[2] = item[2]*item[1]
167
```

```
168 # Sums the total of workers in the city and then calculates the cumulative proportions for each neighbourhood (
        similarly as before).
169 for i in range(2, len(cityWorkProportions)):
170     cityWorkProportions[1] += cityWorkProportions[i]
171 for i in range(2, len(cityWorkProportions)):
172     cityWorkProportions[i] = cityWorkProportions[i]/cityWorkProportions[1]
173 for i in range(3, len(cityWorkProportions)):
174     cityWorkProportions[i] += cityWorkProportions[i-1]
175
176 # NLines carries the neighbourhoods IDs.
177 N.readline()
178 NLines = N.readlines()
179 cnt = 0
180
181 # Write the header.
182 O.write('id\tage\thomeNeighbourhood\tworkNeighbourhood\n')
183
184 # Iterates through neighbourhoods to create the synthetic population according to proportions.
185 for item in nProportions:
186     homeNeighbourhood = item[0]
187     population = item[1]
188
189     for p in range(population):
190         # Random number to match the age group proportion.
191         rnd1 = random.random()
192         for ageRange in range(2,len(item)):
193             if rnd1 <= float(item[ageRange]):
194                 break
195         # Finds the correspondent age group.
196         for i in range(10):
197             if i == (ageRange-2):
198                 if i == 0:
199                     age = "00 to 05"
200                 elif i == 1:
201                     age = "06 to 09"
202                 elif i == 2:
203                     age = "10 to 14"
204                 elif i == 3:
205                     age = "15 to 17"
206                 elif i == 4:
207                     age = "18 to 24"
208                 elif i == 5:
209                     age = "25 to 34"
210                 elif i == 6:
211                     age = "35 to 44"
212                 elif i == 7:
213                     age = "45 to 59"
214                 elif i == 8:
215                     age = "60 to 64"
216                 else:
217                     age = "64 or more"
218
219         # 47,4% of the city's population works according to external data (source: demographic information from the
                city's OD report).
220         workNeighbourhood = -1
221         # Checks if it will be a worker.
222         rnd2 = random.random()
223         if rnd2 > 0.474398868:
224             # Finds a work neighbourhood for the person from it's home neighbourhood worker distribution proportions.
225             rnd3 = random.random()
```

```
226                 found = False
227                 for workItem in wProportions:
228                     if workItem[0] == homeNeighbourhood:
229                         for workRange in range(2,len(workItem)):
230                             if rnd3 <= float(workItem[workRange]):
231                                 found = True
232                                 break
233                         break
234
235                 # There aren't available individual data for worker distribution for all neighbourhoods,
236                 # for these cases we use the city's general proportion.
237                 if found == False:
238                     for workRange in range(2,len(cityWorkProportions)):
239                         if rnd3 <= float(workItem[workRange]):
240                             break
241                 workNeighbourhood = nNames[workRange-2]
242
243             # Finds the worker's work and home neighbourhoods' ID. If he is not a worker, then the work neighbourhood's
                    ID is "000".
244             for nItem in NLines:
245                 lineS = nItem.split("\t")
246                 if lineS[1] == item[0]:
247                     idN = lineS[0]
248                 if lineS[1] == workNeighbourhood:
249                     idW = lineS[0]
250                 elif workNeighbourhood == -1:
251                     idW = "000"
252
253             # The ID of the person is composed by his home neighbourhood's ID plus his work neighbourhoods ID plus the
                    counter.
254             idP = str(idN) + idW + str(cnt)
255
256             # Writes the person to the output file.
257             O.write(idP + "\t" + age + "\t" + homeNeighbourhood + "\t" + str(workNeighbourhood) + "\n")
258
259             # Updates the counter.
260             cnt = cnt + 1
261
262     print("Finished!")
```

ANEXO E – Tutorial do MATSim adaptado

E.1 CreatePopulation

```
1  public class CreatePopulation {
2
3          private Scenario scenario;
4
5          private String censusFile = " "; // Path to the synthetic census file .
6          private String municipalitiesFile = " "; // Path to the neighbourhood information file.
7
8          private QuadTree<ActivityFacility> homeFacilitiesTree;
9          private QuadTree<ActivityFacility> workFacilitiesTree;
10
11         private TreeMap<String, Coord> municipalityCentroids = new TreeMap<>();
12         private Random random = new Random(3838494);
13
14         private ObjectAttributes personHomeAndWorkLocations = new ObjectAttributes();
15         private final static Logger log = Logger.getLogger(CreatePopulation.class);
16
17         //
               ------------------------------------------------------------------------
18
19         public void run(Scenario scenario) {
20                 this.scenario = scenario;
21                 this.init();
22                 this.populationCreation();
23         }
24
25         private void init() {
26                 /*
27                  * Build quad trees for assigning home and work locations
28                  */
29                 this.homeFacilitiesTree = this.createActivitiesTree("home",
30                                 this.scenario);
31                 this.workFacilitiesTree = this.createActivitiesTree("work",
32                                 this.scenario);
33
34                 this.readMunicipalities();
35         }
36
37         private void populationCreation() {
38                 /*
39                  * For convenience and code readability store population and population
40                  * factory in a local variable
41                  */
42                 Population population = this.scenario.getPopulation();
43                 PopulationFactory populationFactory = population.getFactory();
44
45                 /*
46                  * Read the census file Create the persons and add the
47                  * socio-demographics
48                  */
49                 try {
```

```
50                  BufferedReader bufferedReader = new BufferedReader(new FileReader(
51                          this.censusFile));
52                  String line = bufferedReader.readLine(); // skip header
53
54                  int index_personId = 0;
55                  int index_age = 1;
56
57                  while ((line = bufferedReader.readLine()) != null) {
58                      String parts[] = line.split("\t");
59
60                      /*
61                       * Create a person and add it to the population
62                       * Adds only people that drives by car to the population.
63                       * This information was previously found elsewhere.
64                       * 36.5% of the population that doesn't work uses car in it's routine by car.
65                       * 28.7% of the employed population uses car.
66                       */
67                      if (parts[index_personId].substring(3, 6).equals("000")) {
68                          if (random.nextFloat() > 0.365715823466093) {continue;}
69                          }
70                      else {
71                          if (random.nextFloat() > 0.286821705426357) {continue;}
72                          }
73
74                      Person person = populationFactory.createPerson(this.scenario
75                              .createId(parts[index_personId]));
76                      ((PersonImpl) person)
77                              .setAge(Integer.parseInt(parts[index_age].substring(0, 2)));
78
79                      // "000" is the value for non-worker in person's ID in the digits 3 to 6.
80                      // Other numbers relate to their work neighbourhoods.
81                      boolean employed = true;
82                      if (parts[index_personId].substring(3, 6).equals("000"))
83                          employed = false;
84                      ((PersonImpl) person).setEmployed(employed);
85                      population.addPerson(person);
86
87                      // The three first digits of the person's ID carry it's home neighbourhood location.
88                      String homeMunicipality = parts[index_personId].substring(0, 3);
89                      ActivityFacility homeFacility = this
90                              .getFacility("home", homeMunicipality);
91                      personHomeAndWorkLocations.putAttribute(person.getId()
92                              .toString(), "home", homeFacility);
93
94                      if (employed) {
95                          /*
96                           * Assign a work location and buffer it somewhere. This
97                           * could also be done in the persons knowledge. But we use
98                           * ObjectAttributes here.
99                           */
100                         String workMunicipality = parts[index_personId].substring(3, 6);
101                         ActivityFacility workFacility = this
102                                 .getFacility("work", workMunicipality);
103                         personHomeAndWorkLocations.putAttribute(person.getId()
104                                 .toString(), "work", workFacility);
105                     }
106                 }
107                 bufferedReader.close();
108
109             } // end try
```

```
110             catch (IOException e) {
111                     e.printStackTrace();
112             }
113     }
114
115     private void readMunicipalities() {
116             try {
117                     BufferedReader bufferedReader = new BufferedReader(new FileReader(
118                                     this.municipalitiesFile));
119                     String line = bufferedReader.readLine(); // skip header
120
121                     while ((line = bufferedReader.readLine()) != null) {
122                             String parts[] = line.split("\t");
123
124                             String id = parts[0];
125
126                             Coord coord = new CoordImpl(Double.parseDouble(parts[3]),
127                                             Double.parseDouble(parts[4]));
128                             this.municipalityCentroids.put(id, coord);
129                             ;
130                     }
131                     bufferedReader.close();
132
133             } // end try
134             catch (IOException e) {
135                     e.printStackTrace();
136             }
137     }
138
139     private ActivityFacility getFacility(String type, String municipality) {
140             ArrayList<ActivityFacility> list;
141
142             if (type == "home") {
143                     Coord coord = this.municipalityCentroids.get(municipality);
144                     list = (ArrayList<ActivityFacility>) this.homeFacilitiesTree
145                                     .get(coord.getX(), coord.getY(), 2000);
146
147             } else {
148                     Coord coord = this.municipalityCentroids.get(municipality);
149                     list = (ArrayList<ActivityFacility>) this.workFacilitiesTree
150                                     .get(coord.getX(), coord.getY(), 2000); // 2000 is the aproximate range of the
                                                biggest neighbourhoods in Joinville.
151             }
152
153             int randomIndex = (int) (random.nextFloat() * (list.size() - 1));
154             ActivityFacility chosenFacility = list.get(randomIndex);
155
156             chosenFacility = list.get(randomIndex);
157             return chosenFacility;
158     }
159
160     public Scenario getScenario() {
161             return scenario;
162     }
163
164     public ObjectAttributes getPersonHomeAndWorkLocations() {
165             return personHomeAndWorkLocations;
166     }
167
168     public QuadTree<ActivityFacility> createActivitiesTree(String activityType,
```

```
169                     Scenario scenario) {
170             QuadTree<ActivityFacility> facQuadTree;
171
172             if (activityType.equals("all")) {
173                     facQuadTree = this.buildFacQuadTree(activityType,
174                             ((ScenarioImpl) scenario).getActivityFacilities()
175                                     .getFacilities());
176             } else {
177                     facQuadTree = this.buildFacQuadTree(activityType,
178                             ((ScenarioImpl) scenario).getActivityFacilities()
179                                     .getFacilitiesForActivityType(activityType));
180             }
181             return facQuadTree;
182     }
183
184     private QuadTree<ActivityFacility> buildFacQuadTree(String type,
185                     Map<Id, ? extends ActivityFacility> facilities_of_type) {
186             log.info("building " + type + "facility quad tree");
187             double minx = Double.POSITIVE_INFINITY;
188             double miny = Double.POSITIVE_INFINITY;
189             double maxx = Double.NEGATIVE_INFINITY;
190             double maxy = Double.NEGATIVE_INFINITY;
191
192             for (final ActivityFacility f : facilities_of_type.values()) {
193                     if (f.getCoord().getX() < minx) {
194                             minx = f.getCoord().getX();
195                     }
196                     if (f.getCoord().getY() < miny) {
197                             miny = f.getCoord().getY();
198                     }
199                     if (f.getCoord().getX() > maxx) {
200                             maxx = f.getCoord().getX();
201                     }
202                     if (f.getCoord().getY() > maxy) {
203                             maxy = f.getCoord().getY();
204                     }
205             }
206             minx -= 1.0;
207             miny -= 1.0;
208             maxx += 1.0;
209             maxy += 1.0;
210             System.out.println("          xrange(" + minx + "," + maxx + "); yrange("
211                             + miny + "," + maxy + ")");
212             QuadTree<ActivityFacility> quadtree = new QuadTree<ActivityFacility>(
213                             minx, miny, maxx, maxy);
214             for (final ActivityFacility f : facilities_of_type.values()) {
215                     quadtree.put(f.getCoord().getX(), f.getCoord().getY(), f);
216             }
217             log.info("Quadtree size: " + quadtree.size());
218             return quadtree;
219     }
220 }
```

E.2 CreateDemand

```
1  public class CreateDemand {
2          private Scenario scenario;
3          private Scenario scenarioPUS;
4
```

```
5   private String pusFile = " "; // Path to the treated origin-destination file.
6
7   private ObjectAttributes personHomeAndWorkLocations;
8   private Random random = new Random(3838494);
9
10  private List<Id> pusWorkers = new Vector<Id>();
11  private List<Id> pusNonWorkers = new Vector<Id>();
12
13  private QuadTree<ActivityFacility> shopFacilitiesTree;
14  private QuadTree<ActivityFacility> leisureFacilitiesTree;
15  private QuadTree<ActivityFacility> educationFacilitiesTree;
16  private QuadTree<ActivityFacility> universityFacilitiesTree;
17
18  private final static Logger log = Logger.getLogger(CreateDemand.class);
19
20
21  public void run(Scenario scenario, ObjectAttributes personHomeAndWorkLocations) {
22      this.scenario = scenario;
23      this.scenarioPUS = ScenarioUtils.createScenario(ConfigUtils.createConfig());
24      this.personHomeAndWorkLocations = personHomeAndWorkLocations;
25      this.init();
26      this.createPUSPersons();
27      this.createPUSPlans();
28      this.assignPUSPlansToMATSimPopulation();
29  }
30
31  private void init() {
32      /*
33       * Build quad trees for assigning home and work locations
34       */
35      this.shopFacilitiesTree = this.createActivitiesTree("shop", this.scenario);
36      this.leisureFacilitiesTree = this.createActivitiesTree("leisure", this.scenario);
37      this.educationFacilitiesTree = this.createActivitiesTree("education", this.scenario);
38      this.universityFacilitiesTree = this.createActivitiesTree("university", this.scenario);
39  }
40
41  private void createPUSPersons() {
42      /*
43       * For convenience and code readability store population and population factory in a local variable
44       */
45      Population population = this.scenarioPUS.getPopulation();
46      PopulationFactory populationFactory = population.getFactory();
47
48      /*
49       * Read the PUS file
50       */
51      try {
52          BufferedReader bufferedReader = new BufferedReader(new FileReader(this.pusFile));
53          String line = bufferedReader.readLine(); //skip header
54
55          int index_personId = 0;
56
57          while ((line = bufferedReader.readLine()) != null) {
58              String parts[] = line.split("\t");
59              /*
60               * Create a person and add it to the population
61               */
62              try{
63              Person person = populationFactory.createPerson(this.scenario.createId(parts[
                    index_personId].trim()));
```

```
64                              population.addPerson(person);
65
66                              ((PersonImpl)person).createDesires("desired activity durations");
67                              /*
68                               * Create a day plan and add it to the person
69                               */
70                              Plan plan = populationFactory.createPlan();
71                              person.addPlan(plan);
72                              ((PersonImpl)person).setSelectedPlan(plan);
73                              // The illegal argument will happen when trying to add repeated agents.
74                              // Exists to avoid the necessity of creating a separate file for the PUSpopulation (
                                    and quick adaptation of the code).
75                              }catch (IllegalArgumentException i){}
76                      }
77                      bufferedReader.close();
78              } // end try
79              catch (IOException e) {
80                      e.printStackTrace();
81              }
82      }
83
84      private void createPUSPlans() {
85              /*
86               * For convenience and code readability store population and population factory in a local variable
87               */
88              Population population = this.scenarioPUS.getPopulation();
89              PopulationFactory populationFactory = population.getFactory();
90
91              /*
92               * Read the PUS trips file
93               */
94              try {
95                      BufferedReader bufferedReader = new BufferedReader(new FileReader(this.pusFile));
96                      String line = bufferedReader.readLine(); //skip header
97
98                      int index_personId = 0;
99                      int index_activityDuration = 2;// For first activity of the day (home), this column represents
                            activity end time instead of duration.
100                     int index_mode = 3;
101                     int index_activityType = 1;
102
103                     Id previousPerson = null;
104                     boolean worker = false;
105
106                     while ((line = bufferedReader.readLine()) != null) {
107                             String parts[] = line.split("\t");
108
109                             Id personId = new IdImpl(parts[index_personId]);
110                             Person person = population.getPersons().get(personId);
111
112                             Plan plan = person.getSelectedPlan();
113                             /*
114                              * If a new person is read add a home activity (first origin activity)
115                              * Otherwise add a leg and an activity (destination activity)
116                              */
117                             if (!personId.equals(previousPerson)) {
118                                     Coord coordOrigin = this.scenarioPUS.createCoord(719348.249841,
                                            7087030.903982);
119
120                                     Activity activity = // home
```

```
121                                     populationFactory.createActivityFromCoord("home", coordOrigin);
122
123                                 plan.addActivity(activity);
124
125                                 // define if previous person is a worker or not.
126                                 if (previousPerson != null) {
127                                     if (worker) {
128                                         Person lastWorker = population.getPersons().get(
                                                previousPerson);
129                                         ((PersonImpl) lastWorker).setEmployed(worker);
130                                         this.pusWorkers.add(previousPerson);
131                                     }
132                                     else {
133                                         Person lastNonWorker = population.getPersons().get(
                                                previousPerson);
134                                         ((PersonImpl) lastNonWorker).setEmployed(worker);
135                                         this.pusNonWorkers.add(previousPerson);
136                                     }
137                                 }
138                                 worker = false;
139                             }
140                             else {
141                                 /*
142                                  * Add a leg from previous location to this location with the given mode
143                                  */
144                                 String mode = parts[index_mode];
145                                 plan.addLeg(populationFactory.createLeg(mode));
146
147                                 /*
148                                  * Add activity given its type.
149                                  */
150                                 Coord coordDestination = this.scenarioPUS.createCoord(714236.462315,
                                        7090542.211404);
151
152                                 String activityType = parts[index_activityType].trim();
153                                 if (activityType.startsWith("w")) worker = true;
154
155                                 Activity activity =
156                                     populationFactory.createActivityFromCoord(activityType,
                                            coordDestination);
157
158                                 Double duration = Double.parseDouble(parts[index_activityDuration]);
159                                 // store the desired duration in the persons knowledge
160                                 if (duration != -1.0) ((PersonImpl)person).getDesires().putActivityDuration(
                                        activityType, duration);
161
162                                 plan.addActivity(activity);
163                             }
164                             previousPerson = personId;
165                         }
166                         log.info("Number of workers: " + this.pusWorkers.size());
167                         log.info("Number of non-workers: " + this.pusNonWorkers.size());
168
169                         System.out.println("All elements of pusWorkers: ");
170                         for(int i=0;i<this.pusWorkers.size();i++){
171                             System.out.println(this.pusWorkers.get(i));
172                         }
173                         System.out.println("All elements of pusNonWorkers: ");
174                         for(int i=0;i<this.pusNonWorkers.size();i++){
175                             System.out.println(this.pusNonWorkers.get(i));
```

```
176                     }
177
178                     bufferedReader.close();
179             } // end try
180             catch (IOException e) {
181                     e.printStackTrace();
182             }
183             PopulationWriter populationWriter = new PopulationWriter(this.scenarioPUS.getPopulation(), this.
                    scenario.getNetwork());
184     populationWriter.write(" "); // Path to the origin-destination treated population.
185     }
186
187     private void assignPUSPlansToMATSimPopulation() {
188             /* Iterate through MATSim population and randomly assign a plan from the PUS population.
189              * Adapt the activity locations and the activity end times.
190              */
191             for (Person person : this.scenario.getPopulation().getPersons().values()) {
192                     int i = 0;
193                     Person pusPerson = this.scenarioPUS.getPopulation().getPersons().get(this.pusWorkers.get(0));
194                     String idPusPerson = "999";
195                     String idPerson = person.getId().toString().substring(0,3);
196                     if (((PersonImpl)person).isEmployed()) {
197                             Collections.shuffle(this.pusWorkers, this.random);
198                             // Check if they belong to the same neighbourhood through id (first 3 digits).
199                             while (!idPusPerson.equals(idPerson)) {
200                                     try{
201                                             pusPerson = this.scenarioPUS.getPopulation().getPersons().get(this.
                                                    pusWorkers.get(i));
202                                             idPusPerson = pusPerson.getId().toString().substring(0,3);
203                                     // Fires because for some neighbourhoods there are no OD information. Will
                                            assign a random plan as a solutuion.
204                                     }catch (ArrayIndexOutOfBoundsException q2) {
205                                             pusPerson = this.scenarioPUS.getPopulation().getPersons().get(this.
                                                    pusWorkers.get(0));
206                                             idPusPerson = idPerson;
207                                     }
208                             i++;
209                             }
210
211                             Plan plan = this.adaptAndCopyPlan(person, pusPerson.getSelectedPlan(), true);
212                             person.addPlan(plan);
213                     }
214                     else {
215                             Collections.shuffle(this.pusNonWorkers, this.random);
216                             // Check if they belong to the same neighbourhood through id (first 3 digits).
217                             while (!idPusPerson.equals(idPerson)) {
218                                     try{
219                                             pusPerson = this.scenarioPUS.getPopulation().getPersons().get(this.
                                                    pusNonWorkers.get(i));
220                                             idPusPerson = pusPerson.getId().toString().substring(0,3);
221                                     }catch (ArrayIndexOutOfBoundsException q1) {
222                                             pusPerson = this.scenarioPUS.getPopulation().getPersons().get(this.
                                                    pusNonWorkers.get(0));
223                                             idPusPerson = idPerson;
224                                     }
225                                     i++;
226
227                             }
228
229                             Plan plan = this.adaptAndCopyPlan(person, pusPerson.getSelectedPlan(), false);
```

```
230                             person.addPlan(plan);
231                         }
232                 }
233         }
234
235         private Plan adaptAndCopyPlan(Person person, Plan plan, boolean worker) {
236                 PlanImpl newPlan = new PlanImpl();
237                 newPlan.copyFrom(plan);
238                 /*
239                  * Go through plan and adapt locations and times
240                  */
241                 int counter = 0;
242                 double time = 0.0;
243                 Person pusPerson = plan.getPerson();
244                 Activity previousActivity = null;
245                 String firstType = "";
246                 for (PlanElement pe : newPlan.getPlanElements()) {
247                         if (pe instanceof Activity) {
248                                 ActivityImpl activity = (ActivityImpl)pe;
249                                 ActivityFacility facility ;
250
251                                 // Find facility
252                                 if (activity.getType().startsWith("h")) {
253                                         facility = (ActivityFacility)this.personHomeAndWorkLocations.getAttribute(
                                                person.getId().toString(), "home");
254                                 }
255                                 else if (activity.getType().startsWith("w")) {
256                                         facility = (ActivityFacility)this.personHomeAndWorkLocations.getAttribute(
                                                person.getId().toString(), "work");
257                                 }
258                                 // For agents with study activities and older than 18, looks for a university instead
                                        of regular education facility (schools).
259                                 else if (activity.getType().startsWith("e")) {
260                                         if (((PersonImpl) person).getAge() >= 18){
261                                                 activity.setType("university");
262                                                 facility = this.getRandomLocation(activity, previousActivity.getCoord
                                                        ());
263                                                 activity.setType("education");
264                                         }else facility = this.getRandomLocation(activity, previousActivity.getCoord()
                                                );
265                                 }
266                                 // For leisure.
267                                 else {
268                                         facility = this.getRandomLocation(activity, previousActivity.getCoord());
269                                 }
270
271                                 // Add to the plan
272                                 if (counter == 0) {
273                                         // Considers that every person leaves home around 7 AM with 1.0 sigma in a
                                                Gaussian Curve variation.
274                                         time = 7.0 * 3600.0 + this.randomizeTimes(1.0);
275                                         int suffix = (int)(time / 3600.0);
276                                         if (suffix > 24) suffix = 24;
277                                         activity.setType("home");// + suffix);
278                                         firstType = activity.getType();
279                                         activity.setEndTime(time);
280                                 }
281                                 else if (counter == newPlan.getPlanElements().size() -1) {
282                                         activity.setType(firstType);
283                                 }
```

```
284                                 else {
285                                         double activityDuration = ((PersonImpl)pusPerson).getDesires().
                                                getActivityDuration(activity.getType());
286                                         String prefix = activity.getType().substring(0, 1);
287                                         double sigma;
288                                         // Sets a different sigma value for each type of activity.
289                                         if (prefix.equals("w")) {sigma = 0.25;} // 15 min
290                                         else if (prefix.equals("e")) {sigma = 0.5;} // 30 min
291                                         else if (prefix.equals("s")) {sigma = 0.75;} // 45 min
292                                         else {sigma = 1.0;} // Leisure. 60 min
293                                         time += activityDuration + this.randomizeTimes(sigma);
294                                         if (activityDuration > 86400.0) activityDuration = 86400.0;
295                                         String dur = String.valueOf((int)(activityDuration / 3600.0));
296                                         if (dur.equals("0")) dur = "0.5";
297                                         activity.setType(activity.getType());//.substring(0, 1) + dur);
298                                         activity.setEndTime(time);
299                                 }
300
301                                 activity.setFacilityId(facility.getId());
302                                 activity.setLinkId(facility.getLinkId());
303                                 activity.setCoord(facility.getCoord());
304                                 previousActivity = activity;
305                         }
306                         else {
307                                 Leg leg = (Leg)pe;
308                                 leg.setDepartureTime(time);
309                         }
310                         counter++;
311                 }
312                 return newPlan;
313         }
314
315         private ActivityFacility getRandomLocation(Activity activity, Coord coordPreviousActivity) {
316                 double xCoordCenter = coordPreviousActivity.getX();
317                 double yCoordCenter = coordPreviousActivity.getY();
318                 ArrayList<ActivityFacility> facilities = new ArrayList<ActivityFacility>();
319
320                 if (activity.getType().startsWith("s")) {
321                         double radius = 4000.0;
322                         while (facilities.size() == 0) {
323                                 facilities = (ArrayList<ActivityFacility>) this.shopFacilitiesTree.get(xCoordCenter,
                                        yCoordCenter, radius);
324                                 radius *= 2.0;
325                         }
326                 }
327                 else if (activity.getType().startsWith("l")) {
328                         double radius = 8000.0;
329                         while (facilities.size() == 0) {
330                                 facilities = (ArrayList<ActivityFacility>) this.leisureFacilitiesTree.get(
                                        xCoordCenter, yCoordCenter, radius);
331                                 radius *= 2.0;
332                         }
333                 }
334                 else if (activity.getType().startsWith("e")){
335                         double radius = 3000.0;
336                         while (facilities.size() == 0) {
337                                 facilities = (ArrayList<ActivityFacility>) this.educationFacilitiesTree.get(
                                        xCoordCenter, yCoordCenter, radius);
338                                 radius *= 2.0;
339                         }
```

```
340                 }
341                 else {  // University
342                         double radius = 8000.0;
343                         while ( facilities .size() == 0) {
344                                 facilities  = (ArrayList<ActivityFacility>) this. universityFacilitiesTree .get(
                                        xCoordCenter, yCoordCenter, radius);
345                                 radius *= 2.0;
346                         }
347                 }
348                 int randomIndex = (int)(random.nextFloat() * (facilities.size()));
349                 return  facilities .get(randomIndex);
350         }
351
352         private double randomizeTimes(double sigma) {
353                 return random.nextGaussian() * sigma * 3600.0;
354         }
355
356         public Scenario getScenario() {
357                 return scenario;
358         }
359
360         public QuadTree<ActivityFacility> createActivitiesTree(String activityType, Scenario scenario) {
361                 QuadTree<ActivityFacility> facQuadTree;
362
363                 if (activityType.equals("all")) {
364                         facQuadTree = this.buildFacQuadTree(
365                                         activityType, (Map<Id, ActivityFacility>) ((ScenarioImpl)scenario).
                                                getActivityFacilities().getFacilities());
366                 }
367                 else {
368                         facQuadTree = this.buildFacQuadTree(
369                                 activityType, ((ScenarioImpl)scenario).getActivityFacilities().
                                         getFacilitiesForActivityType(activityType));
370                 }
371                 return facQuadTree;
372         }
373
374         private QuadTree<ActivityFacility> buildFacQuadTree(String type, Map<Id,ActivityFacility>
              facilities_of_type) {
375                 log.info("building " + type + "facility quad tree");
376                 double minx = Double.POSITIVE_INFINITY;
377                 double miny = Double.POSITIVE_INFINITY;
378                 double maxx = Double.NEGATIVE_INFINITY;
379                 double maxy = Double.NEGATIVE_INFINITY;
380
381                 for (final ActivityFacility f : facilities_of_type.values()) {
382                         if (f.getCoord().getX() < minx) { minx = f.getCoord().getX(); }
383                         if (f.getCoord().getY() < miny) { miny = f.getCoord().getY(); }
384                         if (f.getCoord().getX() > maxx) { maxx = f.getCoord().getX(); }
385                         if (f.getCoord().getY() > maxy) { maxy = f.getCoord().getY(); }
386                 }
387                 minx -= 1.0;
388                 miny -= 1.0;
389                 maxx += 1.0;
390                 maxy += 1.0;
391                 System.out.println("          xrange(" + minx + "," + maxx + "); yrange(" + miny + "," + maxy + ")");
392                 QuadTree<ActivityFacility> quadtree = new QuadTree<ActivityFacility>(minx, miny, maxx, maxy);
393                 for (final ActivityFacility f : facilities_of_type.values()) {
394                         quadtree.put(f.getCoord().getX(),f.getCoord().getY(),f);
395                 }
```

```
396                 log.info("Quadtree size: " + quadtree.size());
397                 return quadtree;
398         }
399 }
```

E.3 CreatePopulationAndDemand

```
1  public class CreatePopulationAndDemand {
2
3          private final static Logger log = Logger
4                          .getLogger(CreatePopulationAndDemand.class);
5          private Scenario scenario;
6
7          private String facilitiesFile = " "; // Path to the facilities file .
8          private String networkFile = " "; // Path to the XML network file.
9
10         //
               ------------------------------------------------------------------------------

11         public static void main(String[] args) {
12                 CreatePopulationAndDemand creator = new CreatePopulationAndDemand();
13                 creator.run();
14                 JVControler controler = new JVControler();
15                 //controler.run();
16         }
17
18         private void run() {
19                 this.init();
20                 CreatePopulation populationCreator = new CreatePopulation();
21                 populationCreator.run(this.scenario);
22                 CreateDemand demandCreator = new CreateDemand();
23                 demandCreator.run(this.scenario,
24                                 populationCreator.getPersonHomeAndWorkLocations());
25                 this.write();
26         }
27
28         private void init() {
29                 /*
30                  * Create the scenario
31                  */
32                 Config config = ConfigUtils.createConfig();
33                 this.scenario = ScenarioUtils.createScenario(config);
34                 /*
35                  * Read the network and store it in the scenario
36                  */
37                 new MatsimNetworkReader(this.scenario).readFile(networkFile);
38                 /*
39                  * Read the facilities and store them in the scenario
40                  */
41                 new FacilitiesReaderMatsimV1((ScenarioImpl) this.scenario)
42                                 .readFile(this. facilitiesFile );
43         }
44
45         private void write() {
46                 PopulationWriter populationWriter = new PopulationWriter(
47                                 this.scenario.getPopulation(), this.scenario.getNetwork());
48         populationWriter.write(" "); // Path to the output file .
49                 log.info("Number of persons: "
50                                 + this.scenario.getPopulation().getPersons().size());
```

```
51          }
52  }
```

E.4 CreateFacilities

```
1   public class CreateFacilities {
2
3           private final static Logger log = Logger.getLogger(CreateFacilities.class);
4           private Scenario scenario;
5           private String landUseCensusFile = " "; // Path to the treated land use file .
6
7           public static void main(String[] args) {
8                   CreateFacilities facilitiesCreator = new CreateFacilities();
9                   facilitiesCreator.init();
10                  facilitiesCreator.run();
11                  facilitiesCreator.write();
12                  log.info("Creation finished ##################################");
13          }
14
15          private void init() {
16                  /*
17                   * Create the scenario
18                   */
19                  Config config = ConfigUtils.createConfig();
20                  this.scenario = ScenarioUtils.createScenario(config);
21          }
22
23          private void run() {
24                  /*
25                   * Read the business census for work, shop, leisure and education
26                   * facilities
27                   */
28                  this.readLandUseCensus();
29          }
30
31          private void readLandUseCensus() {
32                  int cnt = 0;
33                  try {
34                          BufferedReader bufferedReader = new BufferedReader(new FileReader(
35                                          this.landUseCensusFile));
36                          String line = bufferedReader.readLine(); // skip header
37
38                          int index_id = 0;
39                          int index_xCoord = 1;
40                          int index_yCoord = 2;
41                          int index_types = 3;
42
43                          while ((line = bufferedReader.readLine()) != null) {
44                                  String parts[] = line.split("\t");
45
46                                  Coord coord = new CoordImpl(
47                                                  Double.parseDouble(parts[index_xCoord]),
48                                                  Double.parseDouble(parts[index_yCoord]));
49
50                                  ActivityFacilityImpl facility = (ActivityFacilityImpl) this.scenario
51                                                  .getActivityFacilities().getFactory()
52                                                  .createActivityFacility(new IdImpl(parts[index_id]), coord);
53                                  ((ScenarioImpl) this.scenario).getActivityFacilities()
54                                                  .addActivityFacility(facility);
```

```
55
56                                  String types[] = parts[index_types].split(",");
57                                  for (int i = 0; i < types.length; i++) {
58                                          this.addActivityOption(facility, types[i]);
59                                  }
60                                  cnt++;
61                          }
62                          bufferedReader.close();
63                  } // end try
64                  catch (IOException e) {
65                          e.printStackTrace();
66                  }
67          }
68
69          private void addActivityOption(ActivityFacility facility, String type) {
70                  ((ActivityFacilityImpl) facility).createActivityOption(type);
71
72                  ActivityOptionImpl actOption = (ActivityOptionImpl) facility
73                                  .getActivityOptions().get(type);
74                  OpeningTimeImpl opentime;
75                  double cap = 0;
76                  if (type.equals("shop")) {
77                          opentime = new OpeningTimeImpl(8.0 * 3600.0, 19.0 * 3600);
78                          cap = 10;
79                  } else if (type.equals("leisure")) {
80                          opentime = new OpeningTimeImpl(8.0 * 3600.0, 21.0 * 3600);
81                          cap = 30;
82                  } else if (type.equals("work")) {
83                          opentime = new OpeningTimeImpl(5.0 * 3600.0, 23.0 * 3600);
84                          cap = 20;
85                  } else if (type.equals("education")) {
86                          opentime = new OpeningTimeImpl(7.5 * 3600.0, 22.5 * 3600);
87                          cap = 100;
88                  } else if (type.equals("university")) {
89                          opentime = new OpeningTimeImpl(7.5 * 3600.0, 22.5 * 3600);
90                          cap = 1000;
91                  // home
92                  } else {
93                          opentime = new OpeningTimeImpl(0.0 * 3600.0, 24.0 * 3600);
94                          cap = 3;
95                  }
96                  actOption.addOpeningTime(opentime);
97                  actOption.setCapacity(cap);
98          }
99
100         public void write() {
101                 new FacilitiesWriter(
102                                 ((ScenarioImpl) this.scenario). getActivityFacilities ())
103                                 .write(" "); // Path to the output file.
104         }
105 }
```

E.5 NetworkToSHP

```
1  public class CreateNetworkSHP {
2
3      public static void main(String[] args) throws Exception {
4
5          Config config = ConfigUtils.createConfig();
```

```
6           config.network().setInputFile(" "); // Path to the XML network file.
7           Scenario scenario = ScenarioUtils.loadScenario(config);
8           Network network = scenario.getNetwork();
9
10          CoordinateReferenceSystem crs = MGC.getCRS("EPSG:31982"); // Sirgas 2000 UTM 22S
11
12          Collection features = new ArrayList();
13          PolylineFeatureFactory linkFactory = new PolylineFeatureFactory.Builder().
14                  setCrs(crs).
15                  setName("link").
16                  addAttribute("ID", String.class).
17                  addAttribute("fromID", String.class).
18                  addAttribute("toID", String.class).
19                  addAttribute("length", Double.class).
20                  addAttribute("type", String.class).
21                  addAttribute("capacity", Double.class).
22                  addAttribute("freespeed", Double.class).
23                  create();
24
25          for (Link link : network.getLinks().values()) {
26              Coordinate fromNodeCoordinate = new Coordinate(link.getFromNode().getCoord().getX(), link.
                    getFromNode().getCoord().getY());
27              Coordinate toNodeCoordinate = new Coordinate(link.getToNode().getCoord().getX(), link.getToNode().
                    getCoord().getY());
28              Coordinate linkCoordinate = new Coordinate(link.getCoord().getX(), link.getCoord().getY());
29              SimpleFeature ft = linkFactory.createPolyline(new Coordinate [] {fromNodeCoordinate, linkCoordinate,
                    toNodeCoordinate},
30                      new Object [] {link.getId().toString(), link.getFromNode().getId().toString(),link.getToNode().
                            getId().toString(), link.getLength(), ((LinkImpl)link).getType(), link.getCapacity(), link.
                            getFreespeed()}, null);
31              features.add(ft);
32          }
33          ShapeFileWriter.writeGeometries(features, "/Users/Bicudo/Dropbox/Faculdade/TCC/MATSim/matsim-0.6.1/
                input/TCC/output_networkJV_08jun_links.shp");
34
35          features = new ArrayList();
36          PointFeatureFactory nodeFactory = new PointFeatureFactory.Builder().
37                  setCrs(crs).
38                  setName("nodes").
39                  addAttribute("ID", String.class).
40                  create();
41
42          for (Node node : network.getNodes().values()) {
43              SimpleFeature ft = nodeFactory.createPoint(node.getCoord(), new Object[] {node.getId().toString()}, null);
44              features.add(ft);
45          }
46          ShapeFileWriter.writeGeometries(features, " "); // Path to the output file.
47      }
48  }
```