Relatório do Trabalho Final

Simulador de Tráfego com Semáforos Inteligentes

Organização (aluno - matrícula):
Diego Feitoza de Oliveira - 412981
Francisco Edvaldo de Oliveira Jr. - 494418

Universidade Federal do Ceará
Disciplina: Sistemas Multi-agentes
Orientação: Prof. Marcos Oliveira
Entrega: 15/07/2022, Quixadá-CE

Sumário

Trabalho Analisado

A proposta do trabalho final foi encontrar um artigo que tratasse do tema de sistema multi-agentes, tentar replicar a parte prática e implementar alguma melhoria.

O artigo escolhido foi um trabalho realizado por ex-alunos da disciplina, Robson Teixeira e Roberta Sousa, junto com os professores Enyo Gonçalves e Marcos de Oliveira, e possui o seguinte título: “An Agent-Based Simulation to Explore Communication in a System to Control Urban Traffic with Smart Traffic Lights”.

Nesse artigo eles exploram a comunicação entre semáforos de uma simulação no simulador de tráfego SUMO, por meio de um sistema multi-agentes implementado com JADE, utilizando como middleware o TraSMAPI, para manipulação dos elementos do SUMO.

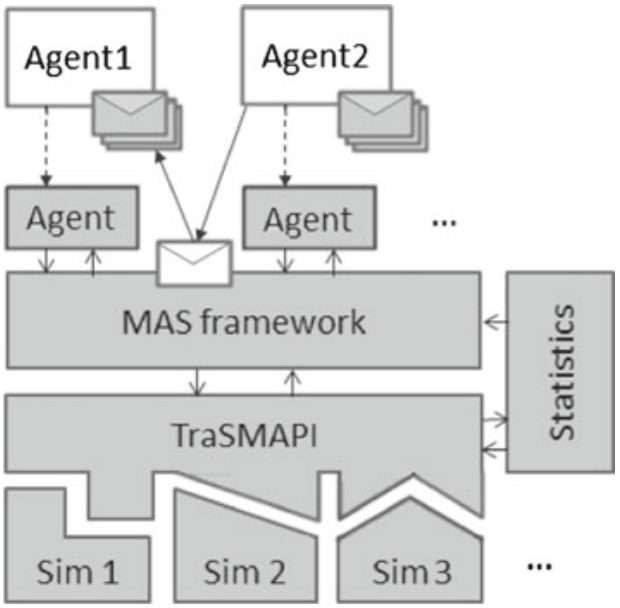

Figura 1 - Exemplo de arquitetura de comunicação entre um sistema multi-agentes e simulações, através do TraSMAPI (fonte).

Artigo escolhido: de Oliveira, M., Teixeira, R. ., Sousa, R., & Tavares Gonçalves, E. J. (2021). An Agent-Based Simulation to Explore Communication in a System to Control Urban Traffic with Smart Traffic Lights. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal, 10(3), 209–225. https://doi.org/10.14201/ADCAIJ2021103209225

Trabalhos Anteriores

O trabalho analisado baseou-se em um trabalho anterior, de 2015, citado abaixo, que propôs o controle de tráfegos por sistema multi-agente.

Artigo base: Junior, G. D., Frozza, R., and Molz, R. F., 2015. Simulação de controle adaptativo de tráfego urbano por meio de sistema multiagentes e com base em dados reais. Revista Brasileira de Computação Aplicada, 7(3):65–81. doi:10.5335/rbca.2015.469.

A proposta deste trabalho foi melhorar o fluxo do tráfego por meio do monitoramento das vias, por meio de sensores conectados aos semáforos.

O algoritmo proposto, imagem abaixo, utiliza os dados coletados para tomar decisões sobre aumentar ou diminuir suas fases de foco de luz verde ou vermelho.

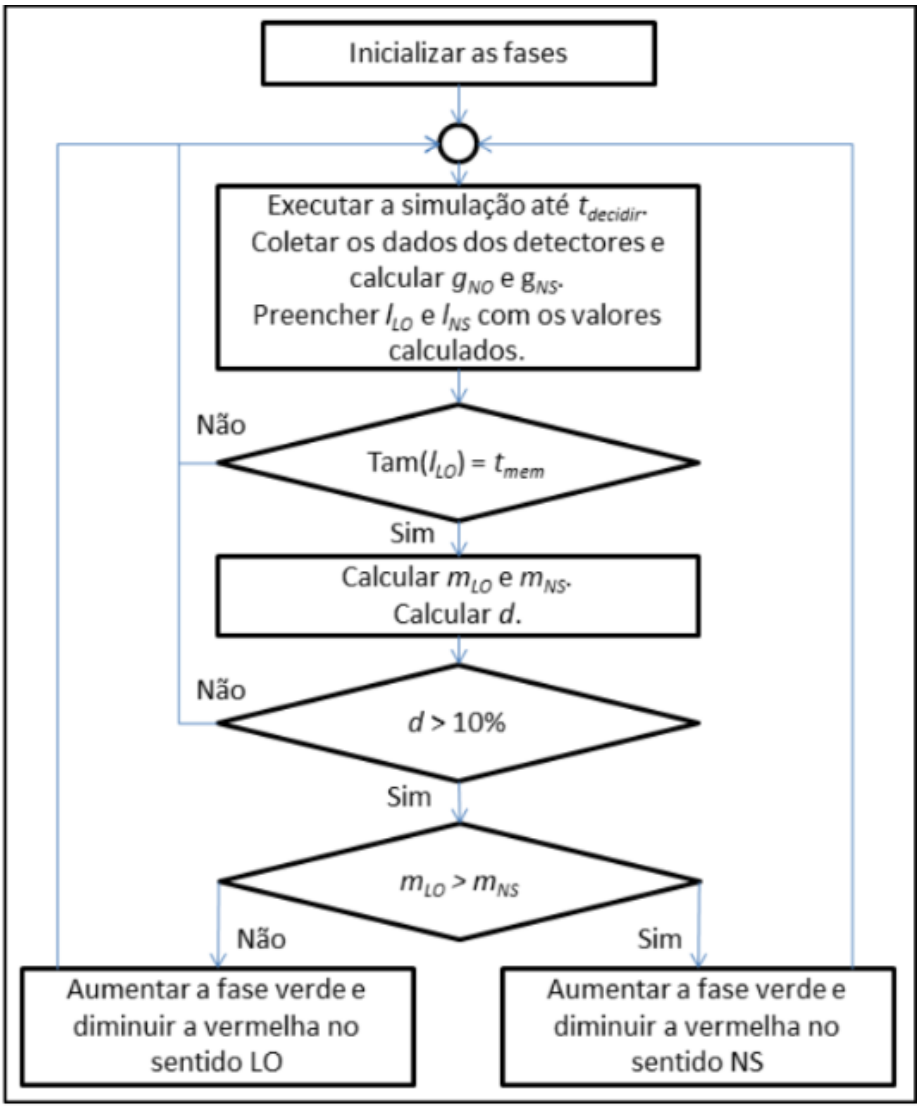

Figura 2: Algoritmo para controle de semáforos inteligentes utilizado no trabalho de Junior, Frozza e Molz (2015).

Melhorias Propostas no Trabalho Analisado

O trabalho do Robson e da Roberta implementa modificações no algoritmo inicial proposto pelo trabalho de 2015. Incluindo a comunicação entre os agentes sobre a quantidade de veículos nas vias monitoradas para tomada de decisão.

Sempre que as condições são atendidas, ou seja, quando há uma variação maior que 10% na média das três últimas médias de velocidade média, na via, entre sensores Leste-Oeste e Norte-Sul, dos sensores vinculados aos semáforos, as fases verde recebem mais um segundo de verde onde a média é menor e mais um segundo de vermelho nas fases onde a média é maior no sentido respectivo.

Com a modificação, quando a variação é menor que 10%, há uma verificação da quantidade de veículos monitorados naquele sentido do semáforo, seja Norte-Sul ou Leste-Oeste, e é aplicada a modificação nas fases dos semáforos para balancear o tempo de verde e vermelho.

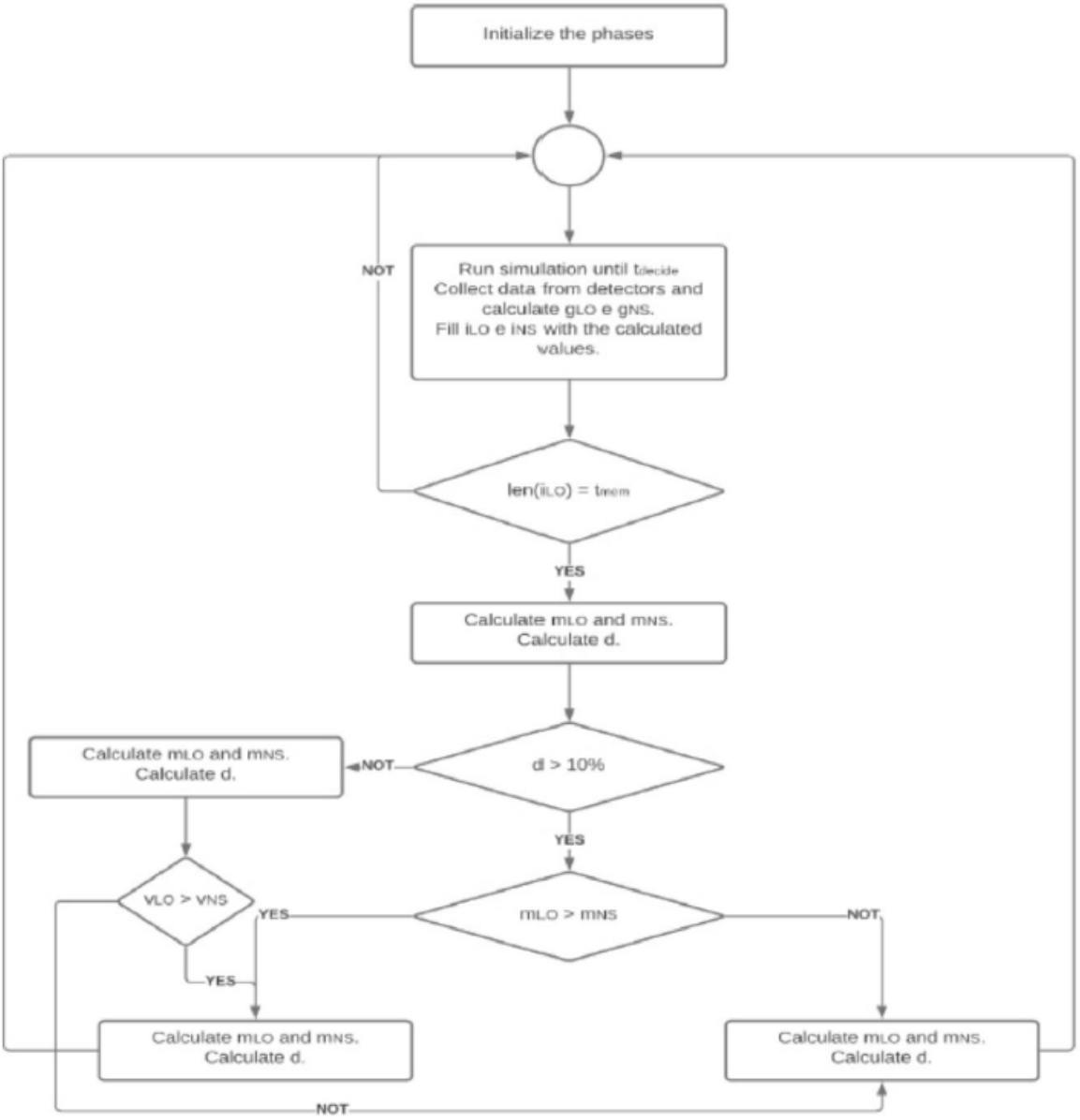

Figura 3: Algoritmo para controle de semáforos inteligentes adaptado.

Analisando o trabalho prático - Configuração

Conseguimos acesso ao trabalho feito pelos ex-alunos por meio do repositório no github, https://github.com/RobsonT/Controle-de-trafego-com-semaforos-inteligentes.

Para o projeto funcionar foi necessário instalar a versão 0.32 do SUMO, pois as versões posteriores a esta não são compatíveis. Baixamos a biblioteca JADE 4.5 e a anexamos como uma biblioteca externa. O TraSMAPI já vem incluído no código. Utilizamos o Java 17.

Parte dos recursos podem ser incluídos como dependências Maven, (SUMO, JADE) apesar de que tivemos dificuldades com a IDE Eclipse.

Após configurar o projeto e executar retornou um erro relacionado ao XML dos arquivos necessário para simulação, então como solução incluímos uma o seguinte trecho “<report><xml-validation value="never"/></report>” no arquivo de configuração do sumo “/network/MAS.sumocfg”. Este arquivo é essencial na configuração da simulação, porém na classe Main também podemos incluir alguns parâmetros de configuração que serão executados na inicialização do simulação.

Analisando o trabalho prático - Código

A pasta “network” contém os arquivos necessários para simulação, incluindo o mapa, os detectores, as rotas e a configuração do sumo.

A classe “Main” contém a inicialização da aplicação, nela podemos configurar os parâmetros do SUMO, inclusive informado os parâmetros de inicialização da simulação. É possível incluir aqui que dados precisamos como saída e local de destino dos arquivos gerados.

Os arquivos gerados pelo SUMO são todos em XML, incluindo os relatórios gerados para análise posterior. Junto com a instalação do sumo há algumas pastas cheias de scripts para auxiliar, entre elas há um conversor de XML para csv, caso seja necessário.

O pacote “agents” contém as classes dos agentes que representam os semáforos, incluindo a lógica de comunicação entre agentes e o algoritmo para tomada de decisão e modificação das fases dos semáforos.

O pacote “agents.sumo” inclui a classe “AgentsManager” que é responsável por receber uma simulação e um container de agentes e converter os semáforos da simulação.

Dentro do pacote “utils” está a classe “E3Detectors” responsável pela leitura do arquivo com os dados dos leitores de tráfego E3. É importante referir aqui o local correto do arquivo caso mude a simulação dentro do projeto. As simulações disponíveis são as das pastas network (Quixadá) e network1 (cidade do projeto base).

Os pacotes do TraSMAPI são responsáveis por fazer comunicação TCP/IP com o SUMO e converter os dados em byte, para conversão em dados para os objetos, fazendo a interface entre os agentes e a simulação. As versões do SUMO posteriores à 0.32 são incompatíveis porque houve mudanças significativas na estrutura dos objetos, exigindo mudança no tamanho em bytes das saídas para cada tipo de dado.

A comunicação TCP/IP atual é feita pelo TraCI e é documentada aqui.

Analisando o trabalho prático - Problemas

Houve uma dificuldade de compreensão do código por conta do nome das variáveis serem resumidas a letras e poucas palavras, tornando difícil acompanhar e compreender a lógica.

O projeto disponível no github não está atualizado conforme o artigo, faltando implementação de parte da lógica de manipulação dos semáforos. Não manipula corretamente as fases do semáforo e não implementa a melhoria de com base na quantidade de veículos das vias, o que faz é a troca de mensagens com as médias de velocidade.

As classes que representam os agentes, TrafficLightAgent e ChangeTrafficLigthStateBehaviour estão unidas em um mesmo arquivo com poucos métodos, gerando um código denso que dificulta a compreensão.

Analisando o trabalho prático - Nossa contribuição

Fizemos um fork do projeto e incluímos melhorias na branch “melhorias”.

Até o momento fizemos as seguintes contribuições:

- Renomeamos as variáveis.
- Separamos a classe ChangeTrafficLigthStateBehaviour da TrafficLightAgent.
- Corrigimos paliativamente o problema de validação do XML.
- Organizamos os arquivos de saída para a pasta “network/output/”.
- Incluímos mais relatórios de saída:
    - Dados sobre o log de erros e alertas do SUMO
    - Dados coletados pelos sensores E3
    - Dados sobre as filas com base no tempo de simulação.
    - Dados relevantes sobre as viagens dos veículos, como velocidade média, tempo de espera, tempo perdido.

Dificuldades

A documentação referente a versão do SUMO utilizada não está disponível e há muitas mudanças desde então, ocasionando limitações para evluir o código.

É muito difícil encontrar documentação ou outros projetos em Java, incluindo multi-agentes, que faça integração com o SUMO. A maioria está concentrada em Python, incluindo exemplos e tutoriais do SUMO. O próprio projeto do SUMO e seus scripts (p.ex. tools) são desenvolvidos em Python.

Desde a versão 0.32 já tiveram muitas bibliotecas para Java, mas praticamente todas descontinuadas. Restando a libsumo e libtraci, que possuem pouca documentação disponível.

A versão utilizada no projeto se tornou obsoleta, inclusive as melhorias implementadas, visto que o SUMO atual está bem mais eficiente e traz diversas melhorias implementadas diretamente nos semáforos.

Tentamos atualizar o TraSMAPI, mas é uma tarefa complexa, por conta da conversão em bytes na comunicação TCP/IP e a exigência de compreender detalhadamente o funcionamento do SUMO.

Outras tentativas foram utilizar o TraaS (github, documentação) e tentar atualizar para as bibliotecas atuais, como libsumo e libtraci, mas por falta de documentação e tempo não conseguimos fazer a conversão.

Trabalhos Futuros - Possíveis Caminhos

Caso queira seguir com java, pode começar com esses dois TCCs, são os mais recentes que utilizam a combinação do SUMO com JADE. Esse de 2020 utiliza o TraaS como middleware. Apesar de o código não estar disponível no github, ele é detalhado no artigo.

Santos, Víctor Muniz dos. *Interligação Do Framework JADE Com a Ferramenta SUMO Em Um Domínio De Smart Parking Baseado Em Agentes*. 2020.

https://oasisbr.ibict.br/vufind/Record/UTFPR-12_a6bbf9a1cfcd6954fb247993bbd3045f

DUCHEIKO, Felipe Felix. Implementação de um sistema multi-agente com mecanismo de negociação descentralizado para um estacionamento inteligente. 2019. 93 f. Trabalho de Conclusão de Curso (Ciência da Computação) - Universidade Tecnológica Federal do Paraná, Ponta Grossa, 2019.

https://repositorio.utfpr.edu.br/jspui/bitstream/1/15996/1/PG_COCIC_2019_1_06.pdf

Outra opção seria visitar diretamente o repositório, https://github.com/laca-is, do laboratório de pesquisa da UTFPR, LaCa, pois eles desenvolvem muitos projetos de sistemas multi-agentes.

Porém parece aconselhável fazer projetos em Python, visto que o SUMO é desenvolvido nesta linguagem, trazendo inúmeras vantagens. Inclusive, tornando mais fácil encontrar projetos para usar como ponto de partida.

Trabalhos de Multi-Agente em Python, com SUMO

- Distributed PPO for Traffic Light Control with Multi-Agent RL
- Cooperative Multi-Stage Multi-Goal Multi-Agent Reinforcement Learning (CM3)
- SmartCityBuses

Ponto de partida

Caso queria começar uma simulação, a forma mais prática é baixar o SUMO mais recente e seguir o seguinte tutorial de como criar uma simulação utilizando o OSMWebWizard. Em questão de minutos terá uma simulação personalizada e pronta para testes.