Relatório de Análise dos Arquivos Python do Simulador de Mobilidade Urbana

Data de Elaboração: 2025-06-19

Sumário Executivo e Visão Geral do Projeto

Este relatório detalha o papel, propósito, status de dependência e funcionalidades dos arquivos Python que compõem o sistema de análise e execução do projeto Simulador de Mobilidade Urbana. O projeto, em seu núcleo, parece envolver uma aplicação principal de simulação (presumivelmente desenvolvida em Java, com base nas interações dos scripts) que modela o tráfego urbano. Os scripts Python analisados servem como ferramentas complementares cruciais para processar os resultados da simulação, extrair métricas de desempenho, gerar visualizações e automatizar a execução de múltiplos cenários de simulação para análises comparativas.

Identificamos dois scripts Python principais:
1. `analyze_simulation.py`: Focado na análise detalhada de um único arquivo de log de simulação. Ele extrai dados brutos, calcula uma vasta gama de métricas estatísticas e gera relatórios visuais e tabulares.
2. `run_multiple_simulations.py`: Projetado para orquestrar a execução de várias simulações com diferentes parâmetros (como a duração), coletar os resultados de cada uma e realizar uma análise comparativa, incluindo uma simulação de consumo energético.

Ambos os scripts são fundamentais para a avaliação do desempenho do simulador e para a extração de insights valiosos sobre os cenários de mobilidade urbana modelados. Eles transformam dados brutos de simulação em informações compreensíveis e acionáveis, facilitando a interpretação dos resultados e a tomada de decisões baseada em dados. Este documento fornecerá uma explanação didática de cada script, elucidando sua contribuição individual e coletiva para os objetivos do projeto.

Análise Detalhada dos Scripts Python

Script: `analyze_simulation.py`

Propósito Principal

O script `analyze_simulation.py` tem como objetivo principal realizar uma análise estatística aprofundada dos dados gerados por uma execução do Simulador de Mobilidade Urbana. Ele é projetado para ler um arquivo de log de simulação, extrair informações relevantes sobre o comportamento do tráfego e dos veículos, calcular diversas métricas de desempenho e, por fim, apresentar esses resultados de forma organizada através de tabelas e gráficos. Este script é essencial para transformar os dados brutos da simulação em insights compreensíveis sobre a eficiência da rede de tráfego, o comportamento dos veículos e o impacto dos sistemas de controle, como semáforos.

Funcionalidades Específicas

`parse_simulation_data(filename)`

Esta função é o ponto de partida da análise, responsável por ler e interpretar o conteúdo de um arquivo de log da simulação.
* **Leitura do Arquivo:** Abre e lê o arquivo de log especificado (por exemplo, `simulation_results.txt`).
* **Extração de Dados Básicos:** Utiliza expressões regulares (`re`) para encontrar e extrair informações sumárias da simulação, como:
* Número de vértices carregados no mapa.

* Total de interseções criadas.
* Número máximo de veículos configurado para a simulação.
* Total de veículos efetivamente criados.
* Número de veículos que conseguiram chegar ao seu destino.
* **Extração de Movimentações dos Veículos:** Identifica e processa cada registro de movimentação de veículo no log. Para cada movimento, extrai:
* `tempo` : O instante da simulação em que o movimento ocorreu.
* `veiculo_id` : O identificador único do veículo.
* `origem` : O vértice de origem do movimento.
* `destino` : O vértice de destino do movimento.
* `semaforo_estado` : O estado do semáforo na interseção (ex: VERDE, VERMELHO).
* `parado` : Um booleano indicando se o veículo estava parado (por exemplo, aguardando em um semáforo vermelho).
* **Extração de Caminhos Calculados:** Localiza informações sobre os caminhos (rotas) calculados para os veículos, extraindo:
* `num_vertices` : O número de vértices no caminho.
* `origem` : O vértice inicial do caminho.
* `destino` : O vértice final do caminho.
* **Retorno:** A função retorna um dicionário contendo todos os dados extraídos, organizados em listas e valores numéricos, prontos para serem processados pela função de cálculo de métricas.

`calculate_metrics(data)`

Recebendo os dados parseados pela função anterior, `calculate_metrics` realiza os cálculos estatísticos para derivar métricas de desempenho significativas.
* **Conversão para DataFrame:** Os dados de movimentações e caminhos são convertidos em DataFrames do `pandas` para facilitar a manipulação e análise.
* **Métricas Básicas:** Calcula e armazena métricas gerais da simulação:
* `total_vertices` , `total_intersecoes` , `veiculos_criados` , `veiculos_finalizados` .
* `taxa_conclusao` : Percentual de veículos que chegaram ao destino em relação ao total criado.
* **Duração da Simulação:** Determina a duração total da simulação com base no tempo máximo registrado nas movimentações.
* **Análise por Veículo:** Itera sobre cada veículo único para calcular estatísticas individuais:
* `tempo_viagem` : Tempo total que cada veículo levou desde seu primeiro movimento até o último.
* `tempo_espera` : Tempo total que cada veículo passou parado.
* `movimentos_totais` : Número total de segmentos de via percorridos pelo veículo.
* `eficiencia` : Percentual do tempo de viagem que o veículo passou em movimento.
* Essas estatísticas são compiladas em um DataFrame `veiculos_df` .
* **Métricas Agregadas:** Calcula médias a partir das estatísticas individuais dos veículos:
* `tempo_medio_viagem` , `tempo_medio_espera` , `eficiencia_media` .
* **Análise de Semáforos:** Conta a ocorrência de cada estado de semáforo (`distribuicao_semaforos`).
* **Índice de Congestionamento:** Percentual do tempo total de movimentação em que os veículos estavam parados.
* **Fluxo de Veículos:** Analisa o número de movimentações de veículos por unidade de tempo, calculando `fluxo_medio_veiculos` e `fluxo_maximo_veiculos` .
* **Análise de Caminhos:** Calcula `tamanho_medio_caminho` , `tamanho_maximo_caminho` e `tamanho_minimo_caminho` com base nos dados de rotas.
* **Retorno:** Retorna um dicionário contendo todas as métricas calculadas, incluindo os DataFrames intermediários para uso em visualizações.

`create_visualizations(metrics, output_dir)`

Esta função utiliza as métricas calculadas para gerar representações gráficas, facilitando a interpretação dos resultados. Utiliza `matplotlib.pyplot` e `seaborn` para criar os gráficos.

* **Configuração de Estilo:** Aplica um estilo `seaborn-v0_8` e define um tamanho padrão para as figuras.

* **Gráfico de Tempo de Viagem vs. Tempo de Espera:** Um gráfico de dispersão (`scatter plot`) mostrando a relação entre o tempo total de viagem de cada veículo e o tempo que passou esperando. Os pontos são coloridos pela eficiência do veículo.

* **Gráfico de Barras - Métricas por Veículo:** Compara o tempo de viagem e o tempo de espera para cada veículo individualmente.

* **Gráfico de Fluxo de Veículos ao Longo do Tempo:** Um gráfico de linha (`line plot`) que exibe o número de movimentações de veículos em cada unidade de tempo da simulação.

* **Gráfico de Pizza - Distribuição de Estados de Semáforo:** Mostra a proporção de tempo em que os semáforos estiveram em cada estado (VERDE, VERMELHO, AMARELO).

* **Salvamento:** Cada gráfico é salvo como um arquivo PNG no diretório de saída especificado (`output_dir`).

`generate_summary_table(metrics)`

Cria uma tabela resumo formatada contendo as principais métricas da simulação.

* **Estruturação dos Dados:** Organiza um subconjunto das métricas calculadas (como total de vértices, taxa de conclusão, tempo médio de viagem, etc.) em um formato de duas colunas: ‘Métrica’ e ‘Valor’.

* **Criação do DataFrame:** Converte esses dados em um DataFrame do `pandas` .

* **Retorno:** Retorna o DataFrame da tabela resumo.

Função `main()`

A função `main()` é o ponto de entrada do script e coordena a execução de todas as etapas da análise.

1. Invoca `parse_simulation_data` para ler e processar o arquivo de log da simulação (o caminho `/home/ubuntu/workspace/simulation_results.txt` está fixo no código).
2. Passa os dados parseados para `calculate_metrics` para obter as métricas de desempenho.
3. Cria um diretório de resultados (`/home/ubuntu/workspace/results`) se ele não existir.
4. Chama `create_visualizations` para gerar e salvar os gráficos.
5. Chama `generate_summary_table` para criar a tabela resumo.
6. Salva a tabela resumo em um arquivo CSV (`metricas_resumo.csv`) e as métricas detalhadas por veículo em outro CSV (`metricas_por_veiculo.csv`).
7. Imprime a tabela resumo no console e informa ao usuário os nomes dos arquivos gerados e o local onde foram salvos.

Relação com o Simulador de Mobilidade Urbana

O script `analyze_simulation.py` é uma ferramenta de pós-processamento. Ele não interage diretamente com o simulador durante sua execução. Em vez disso, ele consome um artefato produzido pelo simulador: o arquivo de log de texto (ex: `simulation_results.txt`). Sua função é dar sentido aos dados brutos contidos nesse log, traduzindo-os em métricas quantificáveis e visualizações que permitem avaliar o desempenho da simulação e do sistema de mobilidade urbana modelado.

Classificação (Essencial vs. Auxiliar)

Auxiliar. O Simulador de Mobilidade Urbana pode operar e gerar seus logs sem a existência deste script. No entanto, `analyze_simulation.py` é crucial para a interpretação e análise dos resultados da simulação, agregando valor significativo ao projeto ao permitir uma compreensão detalhada do seu comportamento.

Dependências Identificadas

- **Bibliotecas Python:**
  - `re` : Para uso de expressões regulares na extração de dados do log.

    - `pandas`: Para manipulação de dados tabulares e cálculo de estatísticas.
    - `matplotlib.pyplot`: Para a criação de gráficos estáticos.
    - `numpy`: Utilizado implicitamente pelo `pandas` e `matplotlib` para operações numéricas.
    - `collections.defaultdict`: Usado internamente, embora não explicitamente visível na lógica principal das funções descritas.
    - `seaborn`: Para aprimorar a estética dos gráficos e fornecer tipos de plotagem adicionais.
    - `os`: Para manipulação de caminhos de arquivo e criação de diretórios.
- **Arquivos Externos:**
    - Um arquivo de log gerado pelo Simulador de Mobilidade Urbana (ex: `simulation_results.txt`).

Exemplos de Uso e Saídas

- **Execução Típica:** O script é executado via linha de comando, por exemplo: `python analyze_simulation.py`.
- **Entrada:** Espera encontrar um arquivo de log da simulação no caminho `/home/ubuntu/workspace/simulation_results.txt`.
- **Saídas Geradas:**
    - Arquivos CSV no diretório `/home/ubuntu/workspace/results/`:
        - `metricas_resumo.csv`: Tabela com as principais métricas da simulação.
        - `metricas_por_veiculo.csv`: Dados detalhados de desempenho para cada veículo.
    - Arquivos de imagem PNG no diretório `/home/ubuntu/workspace/results/`:
        - `tempo_viagem_vs_espera.png`
        - `metricas_por_veiculo.png`
        - `fluxo_temporal.png`
        - `distribuicao_semaforos.png`
    - Saída no console: Exibição da tabela resumo e uma lista dos arquivos gerados.

Script: `run_multiple_simulations.py`

Propósito Principal

O script `run_multiple_simulations.py` destina-se a automatizar a execução de múltiplas instâncias do Simulador de Mobilidade Urbana, variando parâmetros específicos (como a duração da simulação). Após cada execução, ele coleta os dados, utiliza as funcionalidades de análise do `analyze_simulation.py` para processá-los e, em seguida, compila um relatório comparativo. Adicionalmente, inclui uma funcionalidade para simular o consumo energético com base nas métricas obtidas, oferecendo uma perspectiva mais ampla sobre o desempenho das diferentes configurações de simulação.

Funcionalidades Específicas

`run_simulation(duration=30, output_file=None)`

Esta função é responsável por executar uma única instância do simulador principal (que parece ser uma aplicação Java).
* **Construção do Comando:** Monta o comando necessário para executar o simulador Java. O comando especificado é `java -cp .:libs/json-20240303.jar Main <duration>`, indicando que o simulador é executado a partir de um arquivo `Main.class` (ou `Main.java` compilado), com uma biblioteca JSON no classpath, e recebe a duração da simulação como argumento.
* **Execução do Subprocesso:** Utiliza o módulo `subprocess` do Python para iniciar o processo Java. A execução ocorre no diretório `/home/ubuntu/workspace/Simulador_Mobilidade_Urbana-main/src`.
* **Redirecionamento de Saída:** A saída padrão (`stdout`) e a saída de erro padrão (`stderr`) do simulador Java são redirecionadas para um arquivo de log especificado (por exemplo, `sim_output_{duration}.txt`).

* **Controle de Tempo (Timeout):** Impõe um limite de tempo (60 segundos) para a execução da simulação, prevenindo que processos demorados bloqueiem o script indefinidamente.
* **Tratamento de Erros:** Captura exceções como `subprocess.TimeoutExpired` ou falhas gerais na execução.
* **Retorno:** Retorna o caminho do arquivo de log se a simulação for concluída com sucesso (código de retorno 0); caso contrário, retorna `None`.

`run_comparative_analysis()`

Orquestra a execução de uma série de simulações com diferentes durações e coleta seus resultados para análise.
* **Definição de Cenários:** Define uma lista de durações para testar (ex: `[20, 30, 40, 50]`).
* **Loop de Execução:** Itera sobre cada duração definida:
* Chama `run_simulation()` para executar o simulador com a duração atual.
* Se a simulação for bem-sucedida e o arquivo de log for gerado:
* Utiliza as funções `parse_simulation_data` e `calculate_metrics` (importadas de `analyze_simulation.py`) para extrair e calcular as métricas do arquivo de log recém-criado.
* Adiciona informações de configuração (como a duração configurada) ao dicionário de métricas.
* Armazena as métricas processadas em uma lista de `resultados`.
* **Retorno:** Retorna a lista `resultados`, contendo os dicionários de métricas para cada simulação bem-sucedida.

`create_comparative_report(resultados)`

Gera um relatório tabular que compara as métricas chave entre as diferentes execuções da simulação.
* **Criação de DataFrame Comparativo:** Transforma a lista de `resultados` (obtida de `run_comparative_analysis`) em um DataFrame do `pandas`. Cada linha representa uma simulação, e as colunas representam métricas selecionadas como 'Configuração', 'Duração Real', 'Veículos Criados', 'Taxa Conclusão (%)', 'Tempo Médio Viagem', 'Índice Congestionamento (%)', etc.
* **Salvamento do Relatório:** Salva o DataFrame comparativo como um arquivo CSV (`analise_comparativa.csv`) no diretório `/home/ubuntu/workspace/results`.
* **Exibição no Console:** Imprime a tabela comparativa formatada no console.
* **Cálculo de Estatísticas Agregadas:** Calcula e exibe médias gerais para algumas métricas importantes (eficiência média, congestionamento médio, etc.) com base em todos os resultados comparados.
* **Retorno:** Retorna o DataFrame comparativo.

`simulate_energy_consumption(metrics)`

Esta função calcula uma estimativa do consumo energético com base nas métricas de uma simulação.
* **Parâmetros de Consumo (Simulados):** Utiliza valores fixos e hipotéticos para o consumo energético:
* `CONSUMO_POR_MOVIMENTO`: Energia gasta por cada movimento de veículo.
* `CONSUMO_PARADO`: Energia gasta por unidade de tempo que um veículo permanece parado.
* `CONSUMO_BASE`: Consumo base por unidade de tempo de viagem de um veículo.
* **Cálculo por Veículo:** Se dados de veículos (`veiculos_stats`) estiverem disponíveis nas métricas, itera sobre cada veículo:
* Calcula o consumo devido aos movimentos, ao tempo parado e ao tempo total de viagem.
* Soma esses valores para obter o consumo total da simulação.
* **Métricas Energéticas:** Calcula:
* `consumo_total_kwh`: Consumo total de energia.
* `consumo_por_veiculo`: Consumo médio por veículo criado.
* `eficiencia_energetica`: Uma métrica de eficiência baseada no índice de congestionamento.
* **Retorno:** Retorna um dicionário com as métricas de consumo energético calculadas.

Função `main()`

Coordena a execução de todo o processo de análise comparativa e simulação de consumo energético.
1. Chama `run_comparative_analysis()` para executar as múltiplas simulações e coletar seus resultados.

2. Se houver resultados:
* Chama `create_comparative_report()` para gerar e salvar o relatório comparativo em CSV e exibi-lo.
* Itera sobre os resultados de cada simulação, chama `simulate_energy_consumption()` para cada um e imprime a análise de consumo energético no console.
3. Informa ao usuário sobre a conclusão da análise e os arquivos gerados.
4. Se nenhuma simulação for bem-sucedida, informa o usuário.

Relação com o Simulador de Mobilidade Urbana

Este script atua como um orquestrador ou um "gerenciador de experimentos" para o Simulador de Mobilidade Urbana principal (a aplicação Java). Ele invoca diretamente o simulador múltiplas vezes com diferentes parâmetros.

Além disso, ele depende funcionalmente do `analyze_simulation.py` para processar os arquivos de log individuais gerados por cada execução do simulador Java. Portanto, ele se posiciona entre o usuário (que deseja rodar múltiplos cenários) e o simulador, automatizando um processo que seria manual e repetitivo.

Classificação (Essencial vs. Auxiliar)

Auxiliar. Similar ao `analyze_simulation.py`, o simulador principal pode ser executado independentemente deste script. No entanto, `run_multiple_simulations.py` oferece uma capacidade avançada de automação para testes de cenários e análises comparativas, o que é extremamente útil para pesquisa e desenvolvimento iterativo do modelo de simulação ou para estudos de impacto de diferentes configurações.

Dependências Identificadas

- **Bibliotecas Python:**
  - `subprocess`: Para executar e gerenciar o processo do simulador Java.
  - `os`: Para manipulação de caminhos e verificação da existência de arquivos.
  - `pandas`: Para criar e manipular o DataFrame do relatório comparativo.
  - `json`: Importado, mas não utilizado diretamente no código fornecido; pode ser uma dependência do simulador Java que ele invoca ou planejado para uso futuro.
- **Módulos do Projeto:**
  - Funções `parse_simulation_data` e `calculate_metrics` do script `analyze_simulation.py`.
- **Componentes Externos:**
  - O Simulador de Mobilidade Urbana em Java: Requer que `Main.class` (ou equivalente compilado) e `libs/json-20240303.jar` estejam acessíveis no caminho `/home/ubuntu/workspace/Simulador_Mobilidade_Urbana-main/src`.
  - Arquivos de log gerados por cada execução do simulador Java (ex: `sim_output_20.txt`).

Exemplos de Uso e Saídas

- **Execução Típica:** O script é executado via linha de comando, por exemplo: `python run_multiple_simulations.py`.
- **Entrada:** Nenhuma entrada direta via argumento de linha de comando é mostrada, mas ele opera com base nas durações pré-definidas internamente.
- **Saídas Geradas:**
  - Arquivos de log individuais para cada simulação no diretório `/home/ubuntu/workspace/` (ex: `sim_output_20.txt`, `sim_output_30.txt`, etc.).
  - Um arquivo CSV de análise comparativa no diretório `/home/ubuntu/workspace/results/`:
    - `analise_comparativa.csv`: Tabela comparando métricas chave entre as diferentes durações de simulação.
  - Saída no console:
    - Progresso da execução das simulações.

- A tabela de análise comparativa.
- Estatísticas agregadas da comparação.
- Análise de consumo energético para cada configuração de simulação.
- Lista dos arquivos gerados.