Universidade Federal de Santa Catarina
Departamento de Informática e Estatística

Gustavo Comiotto Schmitz

OTIMIZAÇÃO DE MALHA VIÁRIA TERRESTRE ATRAVÉS DE ALGORITMO GENÉTICO

Florianópolis
2021

Gustavo Comiotto Schmitz

OTIMIZAÇÃO DE MALHA VIÁRIA TERRESTRE ATRAVÉS DE ALGORITMO GENÉTICO

Trabalho de Conclusão de Curso apresentado à Universidade Federal de Santa Catarina como parte dos requisitos necessários para a obtenção do Grau de Bacharel em Sistemas de Informação.
Orientador: Prof. Dr. Rafael de Santiago

Universidade Federal de Santa Catarina
Departamento de Informática e Estatística

Florianópolis
2021

Gustavo Comiotto Schmitz

OTIMIZAÇÃO DE MALHA VIÁRIA TERRESTRE ATRAVÉS DE ALGORITMO GENÉTICO

Trabalho de Conclusão de Curso apresentado à Universidade Federal de Santa Catarina como requisito parcial para a obtenção do grau de Bacharel em Sistemas de Informação.

Comissão Examinadora

Prof. Dr. Rafael de Santiago
Universidade Federal de Santa Catarina
Orientador

Prof. Dr. Alvaro Junio Pereira Franco
Universidade Federal de Santa Catarina

Prof. Dr. Elder Rizzon Santos
Universidade Federal de Santa Catarina

Florianópolis, 24 de março de 2022

Dedico este trabalho a minha família e a todos aqueles que, de alguma forma, auxiliaram para a concretização desta etapa.

Agradecimentos

Primeiramente gostaria de agradecer à minha família. Principalmente aos meus pais que sempre acreditaram em mim, me mostrando que um futuro melhor passa por uma educação de qualidade, sem vocês eu não chegaria aqui. Estendo meu agradecimento a minha esposa Larissa, que me manteve no rumo certo quando muitas vezes me questionei se tudo isso realmente valeria a pena. Aos demais familiares, vocês também fazem parte de tudo isso, muito obrigado! Gostaria de agradecer também a todos os profissionais da educação que fizeram parte da minha vida em algum momento, em especial o Prof. Dr. Rafael de Santiago que foi fundamental na elaboração desse trabalho juntamente com meu colega de bacharelado Nelson Joppi. Gostaria de agradecer a Universidade Federal de Santa Catarina por tornar possível a minha formação para o exercício profissional. Para finalizar um agradecimento especial aos amigos PP, Ivo, Isma, Henrique, Denys e tantos outros feitos durante o curso.

"Viver pra ser melhor, também é um jeito de levar a vida."
(Charlie Brown Jr)

Resumo

Segundo Vianna e Young (2015), a perda total estimada do PIB devido ao trânsito em regiões metropolitanas é de aproximadamente 1,8 % do PIB. Outros fatores ainda podem ser elencados, como danos a saúde dos envolvidos e um aumento no número de acidentes de trânsito com vítimas, fatais ou não. Nesse contexto, este trabalho tem como objetivo a otimização da malha viária através de uma adaptação do modelo computacional de Salman-Alaswad. Este modelo foi adaptado de modo a permitir a inserção e remoção de vias e analisar o comportamento frente as mudanças inseridas. Para isso, efetuou-se: revisão da literatura recente sobre problemas de otimização relacionadas ao projeto de trânsito; levantamento de problemas semelhantes ao proposto na pesquisa; proposição e desenvolvimento do modelo de heurística computacional. Como produto deste trabalho, obteve-se um ranking com os melhores indivíduos por conjunto de parâmetros encontrados durante a fase de análises, o conjunto de parâmetros que apresentou os melhores resultados e o indivíduo com melhor *fitness* encontrado nos experimentos.

Palavras-Chave: Congestionamentos; Regiões metropolitanas; Modelo computacional.

Abstract

According to Vianna e Young (2015), the estimated total loss of GDP due to traffic in metropolitan regions is approximately 1.8% of GDP. Other factors can still be listed, such as damage to the health of those involved and an increase in the number of traffic accidents with victims, fatal or otherwise. In this context, this work aims to the optimization of the road network through an adaptation of the computational model of Salman-Alaswad. This model has been adapted to allow insertion and removal of roads and analyze the behavior in face of the inserted changes. To this end, the following actions had to done: A review of recent literature on optimization problems related to the traffic project; survey of problems similar to the one proposed in the research; proposition and development of the computational heuristic model. As a product of this work, a ranking was obtained with the best individuals per set of parameters found during the analysis phase, the set of parameters that presented the best results and the individual with the best "fitness" found in the experiments.

Keywords: Congestion; Metropolitan regions; Computational model.

Lista de figuras

Lista de tabelas

Sumário

1 Introdução

Grandes engarrafamentos tornaram-se parte da paisagem das grandes metrópoles mundiais. Somadas ao aborrecimento causado pela situação, há também perdas econômicas expressivas. A solução para o problema depende de uma melhor compreensão do uso e da ocupação do solo urbano (MACIEL, 2008). Além do que foi mencionado acima, soma-se o fato de que a frota veicular terrestre brasileira aumentou substancialmente nos últimos anos. De acordo com IBGE (2021), o estado de Santa Catarina conta hoje com um pouco mais de 7,1 milhões de habitantes, no mesmo viés, segundo o DETRAN (2021) a frota catarinense conta com quase 5,5 milhões de veículos. Fazendo uma conta muito simples, onde dividimos o número de veículos pelo total de habitantes percebemos que o estado possui aproximadamente 0,77 veículo por habitante. Tal dado ilustra uma série de problemas, dentre eles a ineficiência do transporte público existente e a falta de alternativas, de preferência sustentáveis, para enfrentar o trânsito, como a falta de ciclovias nas grandes regiões metropolitanas. Isso coloca uma pressão considerável sobre os governos locais e nacionais desenvolver uma infraestrutura de transporte para acompanhar a demanda cada vez maior da população por mobilidade e transporte de mercadorias (SALMAN; ALASWAD, 2018).

Um exemplo de cidade com problemas de tráfego terrestre é Florianópolis. Ela possui indicadores de qualidade de trânsito desfavoráveis, onde trajetos considerados curtos podem facilmente passar de uma hora de duração. Esses mesmos trajetos ainda podem ser mais duradouros quando imprevistos acontecem, como acidentes e chuvas torrenciais. Tal fato é ainda mais grave, quando se trata de uma cidade que possui o turismo como um dos seus pilares financeiros. Com isso, tanto o governo como o comércio, deixam de arrecadar recursos. Por parte do governo, os recursos poderiam ser revertidos em diversas áreas, inclusive na mobilidade urbana.

O Network Design Problem (NDP) visa encontrar o conjunto ideal de vias que devem ser melhorados em uma rede rodoviária, a fim de atingir um determinado objetivo (minimizar congestionamento, poluição ou consumo de energia) (LEBLANC, 1975; POORZAHEDY; TURNQUIST, 1982). Esse problema é endereçado em vários trabalhos, os quais se utilizam de diversos algoritmos e métodos heurísticos.

O modelo NDP apresentado por (SALMAN; ALASWAD, 2018) propõe a redução dos congestionamentos utilizando como modelo computacional uma composição entre a metaheurística de cadeias de Markov e algoritmo genético. Onde otimiza o padrão de tráfego da rede, convertendo seletivamente estradas de mão dupla em estradas de mão única para chegar a um melhor desempenho da rede rodoviária (ou seja, congestionamento total da rede) sem a necessidade de realização de obras de infraestrutura, como construção

de estradas ou adição de faixas (SALMAN; ALASWAD, 2018).

Com isso, este trabalho utiliza o problema dos congenstionamenos caracterizados como NDP utilizando o modelo de Salman e Alaswad (2018) com o objetivo de otimizar o trânsito em diversas regiões, estendendo o modelo para lidar com inclusão e remoção de vias. Para isso, será utilizado a base de dados OpenStreetMap (openstreetmap.org), já o modelo computacional desenvolvido será uma adaptação do modelo de Salman e Alaswad (2018). Feito isso, com posse dos resultados, será efetuada uma análise dos dados obtidos através do modelo computacional desenvolvido.

1.1 Objetivos

Esta seção visa apresentar os objetivos propostos pelo trabalho que será realizado.

1.1.1 Objetivo Geral

Adaptar o modelo de Salman e Alaswad (2018), incluindo a possibilidade de adicionar e remover vias terrestres para otimizar o trânsito de uma determinada região.

1.1.2 Objetivos Específicos

- O1. Efetuar adaptação no modelo matemático de Salman e Alaswad (2018) afim de que novas vias terrestres possam ser inseridas e removidas no projeto da rede;
- O2. Desenvolver metaheurística computacional que contemple o modelo adaptado (produto do subitem acima);
- O3. Analisar os resultados.

1.1.3 Metodologia

O presente trabalho se trata de uma pesquisa quantitativa e exploratória, pois visa adaptar um modelo pré-existente, incluindo novas características, analisando o comportamento do valor de avaliação e da eficiência em tempo computacional do método heurístico.

Para desenvolver a presente pesquisa, pretende-se executar as seguintes etapas:

Primeiramente, efetuou-se um levantamento teórico com intuito de fornecer maior credibilidade a pesquisa. Esta etapa ocorreu com a busca de artigos e livros em alguns repositórios e portais, como o portal CAPES(periodicos.capes.gov.br), o Researchgate(researchgate.net) e o Google Scholar(scholar.google.com.br).

Concluída a primeira etapa, o passo seguinte foi a especificação do modelo. Nesta etapa foi apresentada a adaptação prevista por esse trabalho. Tal adaptação consiste na possibilidade de inserir e remover novas estradas e ruas. Para isso, utilizou-se as

informações extraídas do estudo realizado na etapa anterior, que ajudaram a identificar o que o modelo de (SALMAN; ALASWAD, 2018) e os objetivos dessa pequisa necessitaram.

A próxima etapa consistiu na especificação e desenvolvimento da heurística. Como produto dessa etapa, destaca-se o programa-fonte, escrito na liguaguem de programação Python, do modelo computacional para o modelo adaptado.

A última etapa consistiu na análise dos resultados, na qual esperava-se obter informações sobre o desempenho do método proposto no fluxo de trânsito das instâncias selecionadas.

1.1.4 Estrutura do Texto

Este documento está estruturado em cinco capítulos.

O Capítulo 1, Introdução, apresentou uma visão geral do trabalho, incluído: problematização, objetivos do projeto e metodologia utilizada.

No Capítulo 2, Fundamentação Teórica, possui uma revisão da literatura sobre alguns conceitos abordados nesse trabalho, como Cadeia de Markov, Network Design Problems, Algoritmos Genéticos e Modelos de Usuário .

No Capítulo 3, trabalhos relacionados, é feita a análise de outros trabalhos que abordam um tema similar ao que será apresentado aqui, .

No Capítulo 4, especificação e Métodos, são especificados alguns pontos do trabalho: como a representação de um indivíduo, população inicial, fitness, operadores genéticos e o plano de experimentos.

No Capítulo 5, são apresentadas as Conclusões e trabalhos futuros.

2 Revisão de Literatura

Neste capítulo serão abordados assuntos relativos aos conceitos básicos utilizados neste trabalho. Serão apresentados conceitos estabelecidos na literatura, dando assim um embasamento para a pesquisa.

2.1 Cadeia de Markov

Uma cadeia de Markov é um processo estocástico caracterizado pela propriedade de Markov, que afirma que o estado futuro depende apenas do estado presente, ou em outras palavras, a probabilidade de uma variável estar em um determinado estado depende apenas de seu estado anterior e não do caminho que percorreu para chegar naquele estado anterior (SALMAN; ALASWAD, 2018). As cadeias de Markov são habitualmente definidas através de grafos dirigidos, onde os vértices representam os diferentes estados definidos e as arestas representam a probabilidade de transição entre os estados adjacentes. Mas também podem ser encontradas através de matrizes. As cadeias de Markov são utilizadas em diversas áreas, como química, física, biologia, dentre outras. A Figura 1 ilustra um exemplo de cadeia de Markov simples.

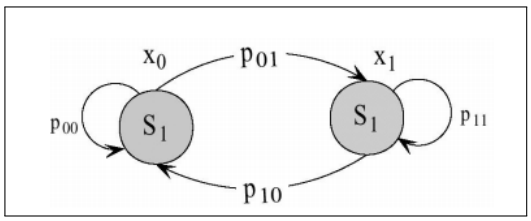


Figura 1 – Exemplo de cadeia de Markov representada por um grafo (GRIGOLETTI, 2015)

2.1.1 Propriedades das cadeias de Markov

Nesta seção serão apresentadas algumas propriedades referentes as cadeias de Markov. Redutibilidade, periodicidade e ergodicidade, respectivamente.

2.1.1.1 Redutibilidade

Para se definir o conceito de redutibilidade, é necessário informar brevemente outros dois conceitos: estados comunicantes e definição de classes. Em uma cadeia de Markov, um estado y é dito comunicante com x quando a partir de y é possível alcançar x e vice-versa. Através do conceito supracitado e a utilização da equação de Chapman-

Kolmogorov, podemos concluir que se o estado x é comunicante com o estado y e o estado y é comunicante com o estado z, então x é comunicante com z. Outra definição é o conceito de classe, onde se dois estados se comunicam entre si, diz-se que eles pertencem a mesma classe.

Com base nas informações acima, uma cadeia de Markov pode ser considerada redutível ou irredutível. Se todos os estados forem comunicantes, logo a cadeia possui apenas uma classe e portanto ela é irredutível. Caso a mesma possua mais de uma classe, ela passa a ser considerada redutível.

2.1.1.2 Periodicidade

Segundo Silva (2017), seja $S = \{s_1, ..., s_k\}$, o período $p(s_i)$ de um estado $s_i \in S$ é definido como:

$$p(s_i) = mdc\{n \geq 1 : (P^n)_{i,i} > 0\} \tag{2.1}$$

Em outras palavras, o período de s_i é o maior divisor comum do conjunto de vezes que a cadeia pode retornar(isto é, tem probabilidade positiva de retorno) a s_i, dado que começamos com $X_0 = s_i$. Se $p(s_i) = 1$, então dizemos que o estado s_i é aperiódico.

2.1.1.3 Ergodicidade

Uma cadeia de Markov é considerada ergódica quando todos os seus estado também forem ergódicos. Um estado é considerado ergódico quando for considerado recorrente, ou seja, uma vez neste estado um retorno a ele é assegurado e aperiódico. Portanto se uma cadeia for irredutível e aperiódica, ela também é consequentemente ergódica.

2.2 Algorítmo Genético

Para Lopes e Takahashi (2011), a computação evolucionária (CE) é um ramo de pesquisa emergente da Inteligência Artificial (IA) que propõe um novo paradigma para a solução de problemas inspirados no livro Seleção Natural (Darwin 1859), que compreende um conjunto de técnicas de busca e otimização inspiradas na evolução natural das espécies. Dado este contexto, os algoritmos genéticos são considerados uma das frentes de pesquisa e estudo da computação evolucionária, justamente por trabalhar com o conceito de populações geradas a partir de operadores genéticos. Tal fato ilustra a semelhança dos algoritmos genéticos com os genes dos seres vivos no processo de evolução natural.

Segundo Darwin (1859), *"Quanto melhor um indivíduo se adaptar ao seu meio ambiente, maior será sua chance de sobreviver e gerar descendentes"*. Esse raciocínio trazido por Charles Darwin, em seu livro *"A origem das espécies"*, está altamente relacionado com o conceito de algoritmos genéticos, principalmente na busca por uma otimização, seja de uma espécie como no caso de Darwin, ou de uma busca computacional.

Com isso, segundo Goldberg (1989), Lacerda, Carvalho e Ludermir (2002) e Gestal (2013) define-se algoritmos genéticos como um método de otimização e buscas inspirado no modelo evolucionista de Darwin onde busca-se evoluir uma população inicial de indivíduos gerados aleatoriamente através dos operadores genéticos(seleção, cruzamento e mutação) afim de alcançar uma boa solução com relação ao escopo em questão. A figura 2 ilustra o funcionamento básico de um algoritmo genético.

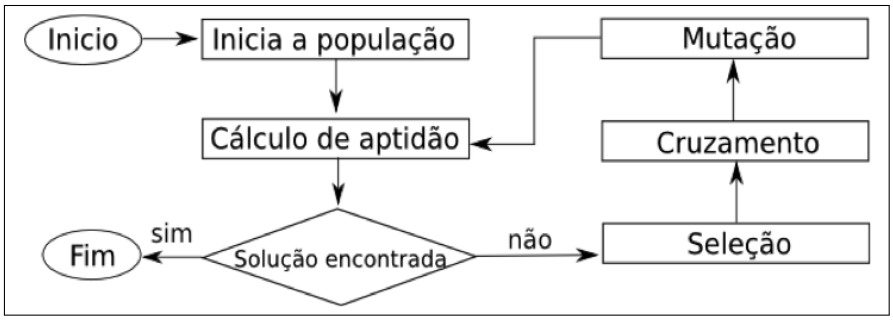


Figura 2 – Exemplo de funcionamento básico de um algoritmo genético.

Os algoritmos genéticos são úteis para auxiliar na resolução de muitos problemas, como na alocação e escalonamento de tarefas, problemas de otimização de aprendizagem de máquina e também em problemas de seleção de rotas que é o contexto que será empregado o algoritmo genético neste trabalho.

2.2.1 Cromossomos e genes

Segundo Barcellos (2000), todos os seres vivos conhecidos são formados por células a partir de um conjunto de instruções contidas no núcleo de todas as células. As instruções citadas são chamadas de genes e estão armazenadas dentro de cada cromossomo que se encontram no núcleo das células. A partir dessa informação podemos realizar uma analogia entre genes e cromossomos com algoritmos genéticos.

Os indivíduos de uma população utilizada em um GA podem facilmente ser comparados com um cromossomo individualmente, pois cada indivíduo possui um conjunto de características, ou genes, que para Barcellos (2000), cada um deles representa uma tentativa de solução, no espaço de soluções possíveis do problema. Já os cromossomos e genes em GA estão diretamente ligados ao fenótipo de cada indivíduo da população, qualquer alteração em algum gene, seja por mutação ou reprodução, pode representar uma melhora ou piora considerável no indivíduo decendente. Tais estruturas são normalmente representadas por uma cadeia de bits, os cromossomos, representados por 0 e 1, cada gene.

2.2.2 Operadores genéticos

Conforme o esquema apresentado na figura 2, para evoluir uma população é necessário que cada indivíduo seja submetido a 3 operadores genéticos, sendo eles: seleção, cruzamento e mutação. Abaixo será feita uma breve explanação sobre os 3 operadores.

2.2.2.1 Seleção

Os algoritmos de seleção são encarregados de escolher quais indivíduos terão oportunidade de se reproduzir e quais não (GESTAL, 2013). Mas isso não quer dizer que os indivíduos taxados como "não aptos" nunca terão nenhuma possibilidade de reproduzir em qualquer geração. Se tal fato ocorresse, teríamos um processo natural de homogenização da população, o que diminuiria a variabilidade disponível para tal população. A operação de seleção pode ser efetuada por duas principais formas, por roleta ou por torneio.

2.2.2.1.1 Seleção por roleta

Neste método, verifica-se a nota de aptidão, ou fitness, de cada indivíduo da geração em que o método será aplicado. Feito isso, é dado a cada indivíduo um "pedaço" proporcional a nota de aptidão de cada um. Então roda-se a agulha da roleta e o indivíduo em cujo pedaço que a agulha parar deve permanecer para a próxima geração. Tal procedimento é repetido até que o número de indivíduos que devem prosseguir seja atingido. A Figura 3 mostra um exemplo prático do método da roleta.

Indivíduo S_i	Aptidão $f(S_i)$	Aptidão Relativa
S_1 10110	2.23	0.14
S_2 11000	7.27	0.47
S_3 11110	1.05	0.07
S_4 01001	3.35	0.21
S_5 00110	1.69	0.11

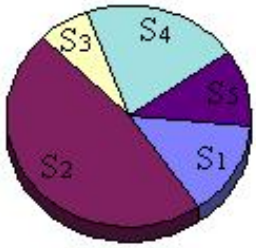


Figura 3 – Exemplo de seleção utilizando roleta (CARVALHO, 2009)

O exemplo da figura acima mostra um exemplo em que temos 5 indivíduos, onde claramente percebe-se que S_2 possui a maior aptidão da geração e S_3 o menor. Com isso, as chances de S_2 avançar à próxima geração é a maior dentre todos os demais.

2.2.2.1.2 Seleção por torneio

Para esse método, determina-se primeiramente o número de indivíduos que serão escolhidos ao acaso para cada rodada. Feito isso, dentre os selecionados, o que possuir a nota de aptidão mais alta está apto a avançar para reproduzir novos descendentes. A Figura 4 ilustra um exemplo prático de seleção por torneio.

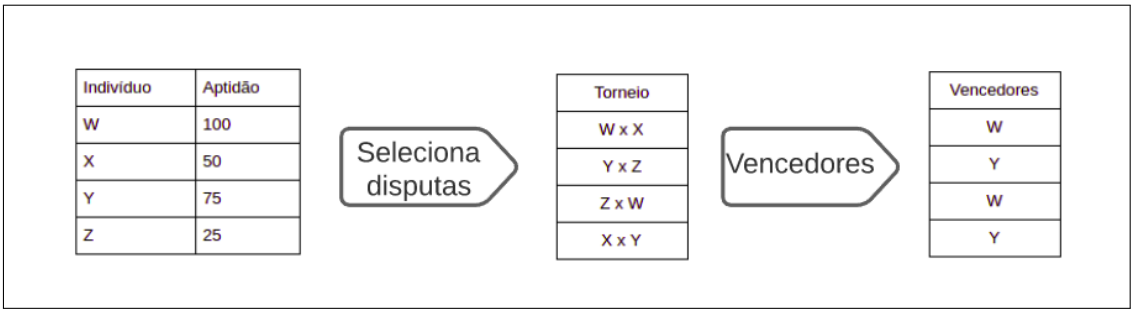


Figura 4 – Exemplo de seleção utilizando torneio.

2.2.2.2 Crossover

Também conhecido como cruzamento, este procedimento ocorre logo após a seleção e utiliza-se da recombinação dos indivíduos selecionados anteriormente para gerar as gerações posteriores. Dentre os tipos de crossover existentes, destacam-se três: um ponto, multiponto e uniforme.

2.2.2.2.1 Crossover em um ponto

Nesse caso, é selecionado um ponto de corte aleatoriamente do cromossomo dos indivíduos progenitores. Esse ponto é o local onde ocorre a troca dos genes para obter novos indivíduos. A figura 5 mostra um exemplo prático do crossover um único ponto.

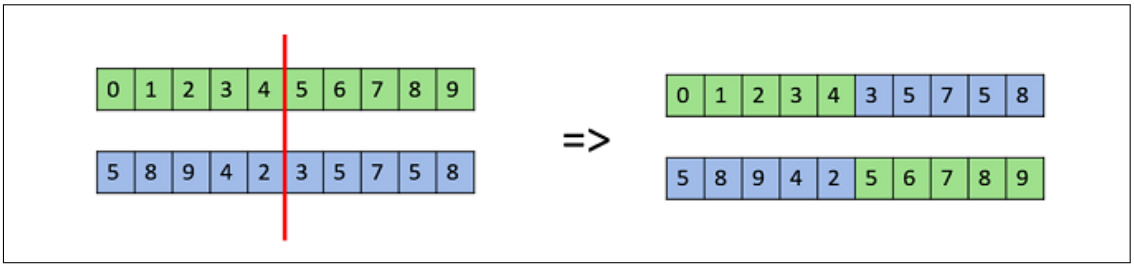


Figura 5 – Exemplo de crossover em um ponto (IT-BRAIN, 2020).

2.2.2.2.2 Crossover multiponto

O crossover multiponto é uma amplificação do crossover de um único ponto, ou seja, possui pelo menos mais um ponto de corte para a troca de genes. A figura 6 ilustra um exemplo do funcionamento do crossover multiponto.

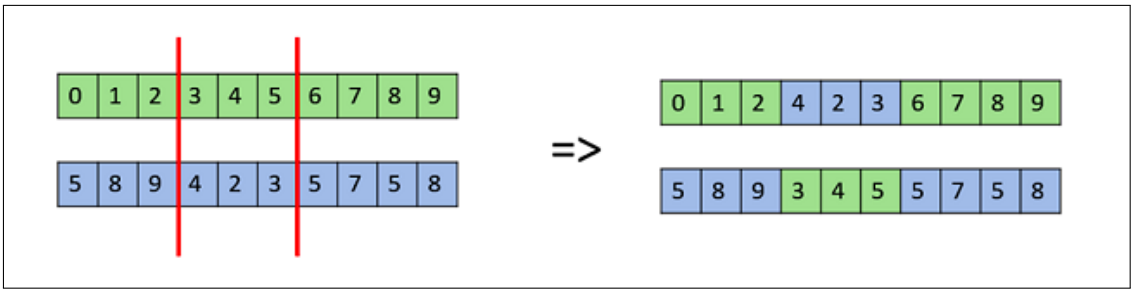


Figura 6 – Exemplo de crossover multiponto (IT-BRAIN, 2020).

2.2.2.2.3 Crossover uniforme

Segundo Gestal (2013), o cruzamento uniforme é uma técnica completamente diferente das vistas até agora. Cada gene na prole tem a mesma probabilidade de pertencer a um ou outro pai. Embora possa ser implementado de muitas maneiras diferentes, a técnica envolve a geração de uma máscara cruzada com valores binários. Se houver um 1 em uma das posições da máscara, o gene localizado nessa posição em um dos descendentes é copiado do primeiro progenitor. Se, por outro lado, houver um 0, o gene usado é do segundo pai. Para produzir a segunda prole, os papéis dos pais são trocados ou a interpretação é trocada com alguns uns e zeros da máscara cruzada. A figura 7 ilustra um exemplo de crossover uniforme.

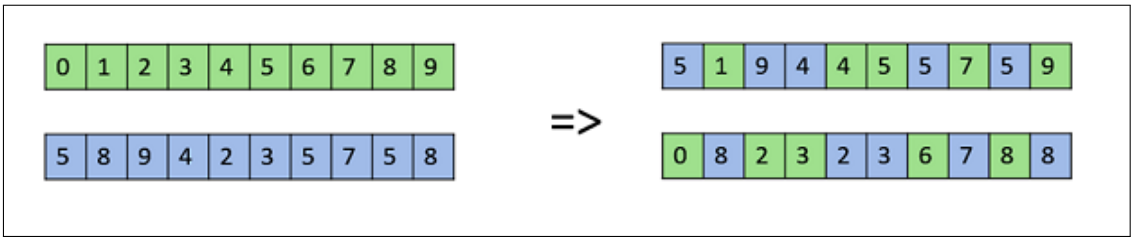


Figura 7 – Exemplo de crossover uniforme. (IT-BRAIN, 2020).

2.2.2.3 Mutação

Existem diversos tipos de mutação, a mais utilizada é a citada em Lacerda, Carvalho e Ludermir (2002), onde o operador de mutação é aplicado juntamente com uma taxa previamente determinada aplicada a cada um dos bits dos filhos gerados, invertendo o

valor de cada gene com o objetivo de aumentar a variabilidade. Recomenda-se que tal taxa não seja tão alta, pois quanto maior ela for, mais randômico se torna a mutação, com isso, aumenta-se a chance dos filhos gerados não serem tão aptos quanto seus progenitores. A figura 8 mostra um exemplo de mutação sendo aplicado.

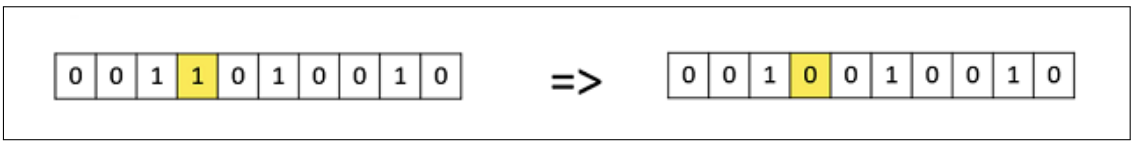


Figura 8 – Exemplo de mutação.

2.3 Road Network Design Problem

Segundo Gallo, D'Acierno e Montella (2010), network design problems consiste em otimizar as características de uma rede viária urbana sem fornecer intervenções infraestruturais (projeto de novas estradas, alargamento de estradas existentes, etc.); podendo envolver toda a rede rodoviária da cidade ou parte dela. Em geral, são considerados problemas de alta complexidade e por isso demandam muitos recursos, principalmente computacionais para sua resolução parcial ou total.

Para Caggiani, Camporeale e Ottomanelli (2017), O NDP é geralmente formulado como um problema de otimização de dois níveis para refletir os diferentes objetivos dos dois tomadores de decisão, que são os usuários da rede e o planejador. Os usuários da rede são livres para escolher suas rotas de forma que seus custos individuais de viagem sejam minimizados, considerando que o planejador visa fazer o melhor uso dos recursos limitados para otimizar o desempenho da rede (por exemplo, reduzindo o congestionamento, minimizando impacto ambiental e maximização de rendimentos), levando em conta o comportamento de escolha de rota dos usuários.

2.3.1 Modelo de usuário

Em um Network Design Problem, tradicionalmente é desenvolvido e exposto um modelo de usuário, que corresponde ao comportamento do mesmo dentro da rede. Seja para um elétron em uma rede elétrica ou para um veículo dentro de uma malha viária, é de suma importância entender o comportamento deles para poder prever e indicar possíveis alternativas. Segundo Duell, Gardner e Waller (2016), os modelos de usuário são capazes de incorporar as diferentes opções de caminhos para cada usuário, podendo ser implementado por exemplo para estimar o tempo de viagem de um veículo, bem como seu consumo.

No modelo proposto por Salman e Alaswad (2018), o comportamento dos usuários nas vias durante a execução do modelo é baseado na densidade máxima de veículos, ou

seja, nos focos de congestionamentos. Diferentemente do comportamento ilustrado nos trabalhos de Caggiani, Camporeale e Ottomanelli (2017) e Poorzahedy e Rouhani (2007), em que baseia-se no tempo de viagem entre dois pontos dentro da rede.

3 Trabalhos Relacionados

Os trabalhos aqui apresentados foram escolhidos após uma pesquisa por artigos relacionados a Road Network Design Problems filtrando pelos mais recentes na plataforma Scopus. Vale a pena mencionar que no início deste trabalho foi determinado que as duas primeiras obras citadas abaixo seriam obrigatoriamente incorporados por serem inspirações para elaboração deste artigo. Portanto, após a realização da pesquisa chegamos a 5 trabalhos, dentre eles removemos o primeiro por não ser mais do nosso interesse. Também removemos o segundo por envolver características muito específicas para o modelo genérico que propomos neste trabalho. Quanto ao terceiro, é um dos trabalhos que optamos descrevê-lo abaixo. Já os últimos dois trabalhos, houve a necessidade de descartá-los devido ao fato de não termos acesso aos mesmos de maneira gratuita.

3.1 Alleviating road network congestion: Traffic pattern optimization using Markov chain traffic assignment

Neste trabalho, os autores propõem uma solução para mitigar a reincidência dos congestionamentos nas vias rodoviárias através de um algoritmo genético com a utilização de cadeias de Markov para avaliação do modelo de ação. Tal otimização é efetuada ao inverter determinadas vias. Com isso, não são necessárias obras de infraestrutura, visto que está acontecendo uma alteração no sentido das faixas e não um acréscimo.

Para realização dessa otimização das vias, os autores optaram por utilizar o critério de densidade máxima de veículos numa estrada como principal indicador de desempenho da rede analisada. Para se mensurar a densidade, os autores utilizam a equação abaixo:

$$D_i = \frac{V\pi_i}{L_i N_i}, \tag{3.1}$$

na qual V representa o número total de veículos na rede rodoviária, π_i representa o vetor de distribuição estacionária, L_i representa o comprimento das estradas e N_i o número de faixas.

Os autores realizaram um experimento utilizando parte da cidade de Abu Dhabi, Emirados Árabes Unidos, onde a rede analisada contava com cerca de 360 estradas e aproximadamente 10.000 veículos transitando na rede. Com isso, eles obtiveram como resultado uma densidade máxima de veículos reduzida de 34,2 para 19,8 veículo/km/via, o que representa uma melhoria de 42%. Para se chegar a tal valor de melhoria foi necessária

a inversão de sentido do fluxo de 24 estradas. A Figuras 9 e 10 ilustram os resultados da execução do modelo

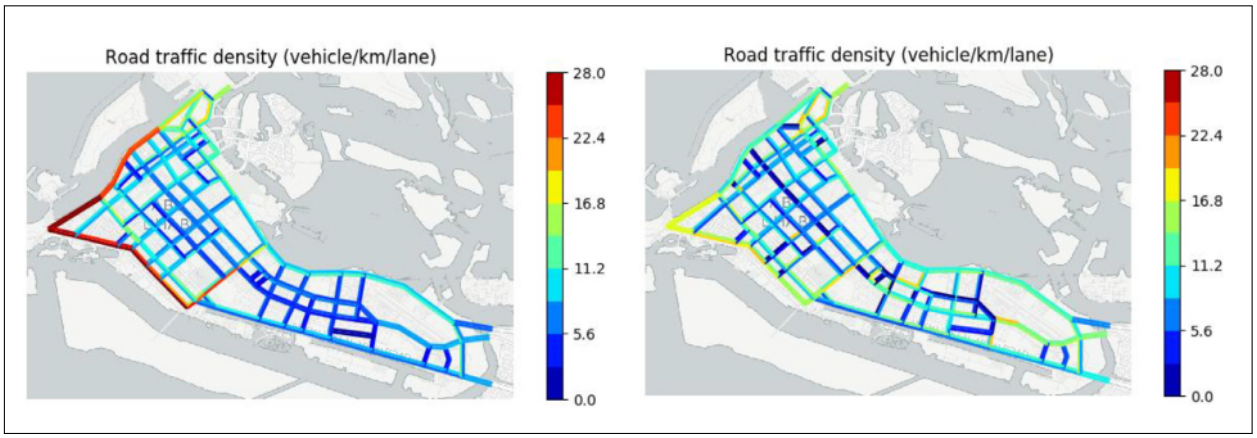


Figura 9 – Densidade das vias antes e depois da execução do modelo. (SALMAN; ALASWAD, 2018)

Figura 10 – As 24 estradas com fluxo invertido. (SALMAN; ALASWAD, 2018)

Outras execuções ainda foram efetuadas pelos autores supondo outras realidades. Por exemplo, partindo do pressuposto que deseja-se alterar o mínimo de vias possível. Com

isso, outros resultados foram obtidos, tanto na eficiência do modelo quanto nas ruas de fato alteradas, bem como sua quantidade. Outro exemplo é a aplicação do modelo em resposta a crises, como um incidente localizado. Novamente resultados totalmente diferentes foram encontrados.

Portanto, será ampliada a ideia (SALMAN; ALASWAD, 2018) para, não apenas inverter o sentido, mas inserir e remover vias no plano viário a ser analisado. De forma que, as novas soluções encontradas possam ser as mais adequadas para uma determinada região.

3.2 Método computacional para otimização do projeto da malha viária de Florianópolis-SC

Seguindo a mesma linha do trabalho anteriormente descrito, os autores também buscam mitigar problemas de congestionamentos em vias urbanas inspirado no trabalho de Salman e Alaswad (2018), porém com cenário e contexto diferentes. Neste trabalho o problema a ser abordado tem como cenário a capital catarinense, Florianópolis (TOMASZEWSKI; SANTIAGO, 2020).

Os autores subdividiram a pesquisa em 6 passos, sendo eles:

- 1 - Identificar as características e gargalos principais de malha viária de Florianópolis (parcial);
- 2 - Enumerar os principais métodos computacionais para o problema de planejamento da malha viária (total);
- 3 - Listar ranqueamento dos métodos computacionais mais adequados ao município de Florianópolis (total);
- 4 - Propor novo método computacional inspirado nos métodos ranqueados e aderentes à realidade do município de Florianópolis (total);
- 5 - Desenvolver método proposto (total);
- 6 - Avaliar o método proposto (total).

Os autores enfrentaram dificuldades na obtenção dos dados para pesquisa, desde uma grande burocracia por parte dos órgãos públicos de trânsito até o alto custo financeiro para obtenção dos mesmos por parte das empresas privadas. Diante desse cenário, optou-se por utilizar a plataforma OpenStreetMap (openStreetMap.org) por se tratar de uma ferramenta gratuita e com ampla documentação.

Quanto ao método proposto, os autores propuseram um algoritmo genético com funções capazes de calcular a probabilidade de um indivíduo passar do ponto a para o ponto

b e também calcular a densidade de cada arco do conjunto de arcos existentes. Para a obtenção dos resultados, foram utilizados dois dos principais operadores genéticos: Mutação e Crossover. Durante a execução da fase de mutação, o algoritmo seleciona aleatoriamente uma via de mão dupla e a torna de mão única. Já na fase de crossover, são selecionados aleatoriamente dois indivíduos para serem os pais e neles é selecionado um ponto p também de maneira aleatória. A partir de p são gerados os indivíduos filhos que darão prosseguimento ao processo.

Para avaliar o método, os autores criaram uma instância artificial com 16 vias, e deixando predeterminado o número de execuções (5), total de gerações em cada execução (50) e com 10 e 25 indivíduos. Os autores mostram dados presentes nas figuras 11 e 12, onde o fitness médio é aferido para populações com 10 e 25 indivíduos, respectivamente:

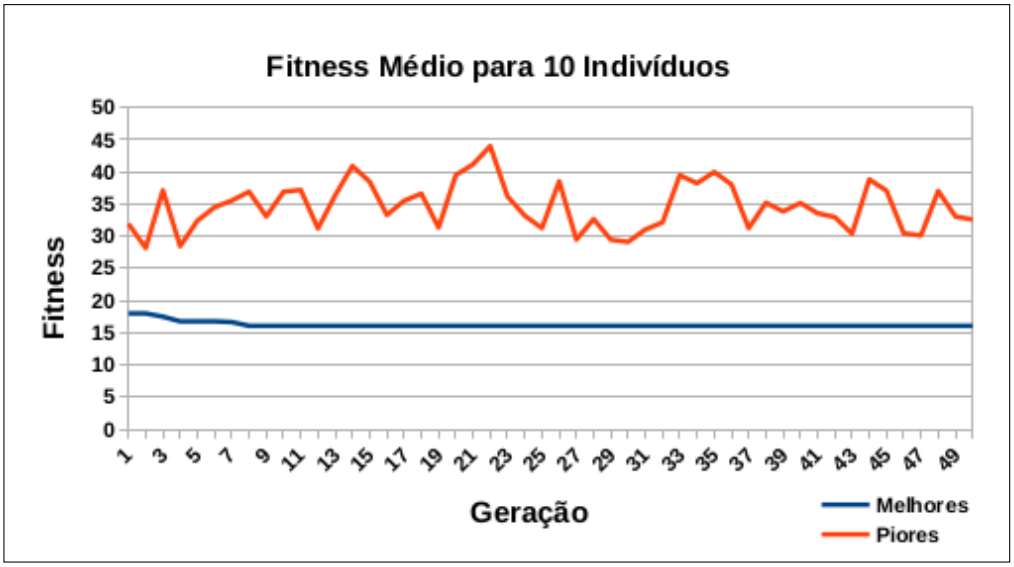


Figura 11 – Fitness médio para população de 10 indivíduos

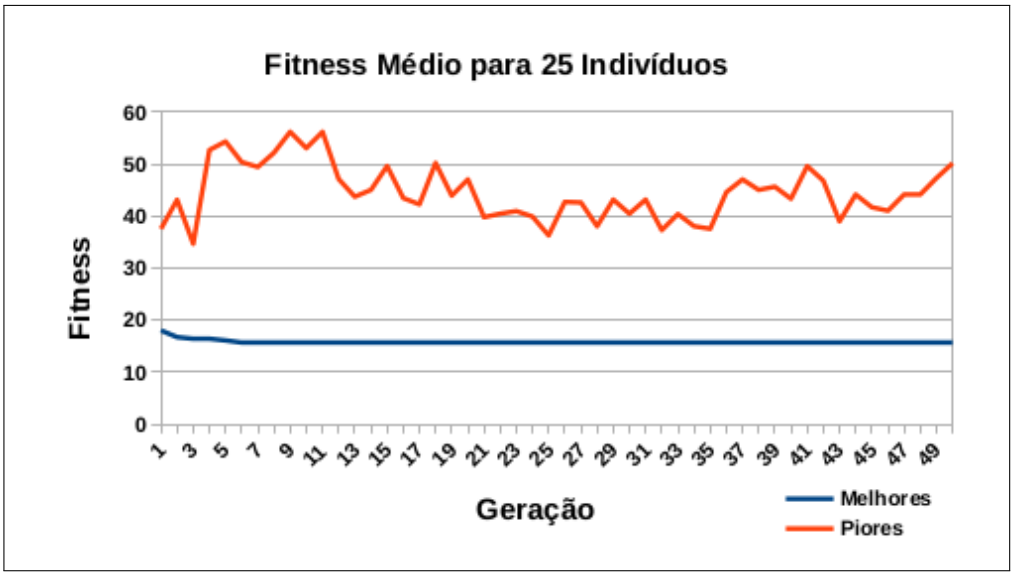


Figura 12 – Fitness médio para população de 25 indivíduos

Segundo os autores, tais dados ilustram uma estabilidade nos melhores indivíduos gerados a partir de determinada geração, para 25 indivíduos verificou-se a partir da sexta, já para 10 indivíduos a partir da nona. Tal fato comprova a influência da quantidade de indivíduos na determinação dos resultados. Já a figura 13 mostra o fitness médio dos melhores usuários em ambas as execuções:

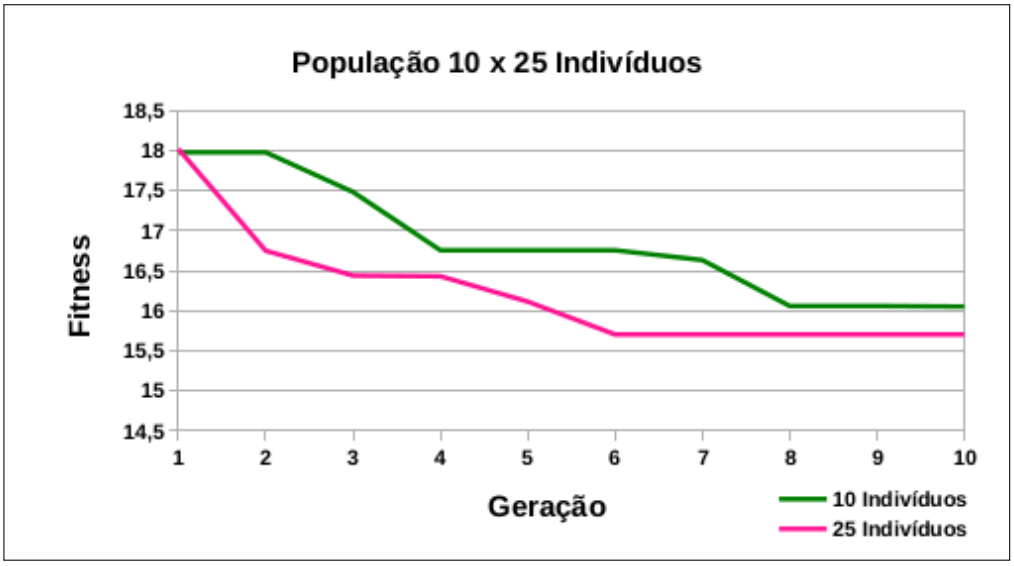


Figura 13 – Fitness médio por geração em populações com 10 e 25 indivíduos

Com base no gráfico acima, ficou evidente para os autores que com uma quantidade maior de instâncias é possível obter resultados melhores. O método foi desenvolvido na linguagem de programação Python e se encontra num repositório público no Github (https://github.com/Seis/NDP2020-Tomaszewski-Santiago).

Assim como descrito em 3.1, será ampliada a ideia de (TOMASZEWSKI; SANTIAGO, 2020) para contemplar também a inserção e remoção de vias no plano viário a ser analisado para este trabalho.

3.3 A general methodology for reducing computing times of road network design algorithms

Neste trabalho os autores têm como proposta a redução do tempo computacional para um Road Network Design Problem (RNDP) voltado para otimização rodoviária para zona rural, mais especificamente para a cidade de Vilnius (Lituânia). Para essa realizar tal fato, a cada iteração do algoritmo de solução de rede, exceto o primeiro, os fluxos de tráfego de equilíbrio obtidos no final da iteração anterior são utilizados como iniciais no próximo procedimento de atribuição. Para os autores, a abordagem exposta só é viável se a topologia da rede rodoviária permanecer inalterada ao longo das iterações.

Os autores realizaram um experimento a fim de comparar a abordagem proposta com a tradicional, eles subdividiram a área do condado de Vilnius em 235 zonas de tráfego, onde a cidade de Vilnius foi subdividida utilizando uma grade de 2,5 km, já o restante da área com uma grade de 10,0 km, conforme a figura 14.

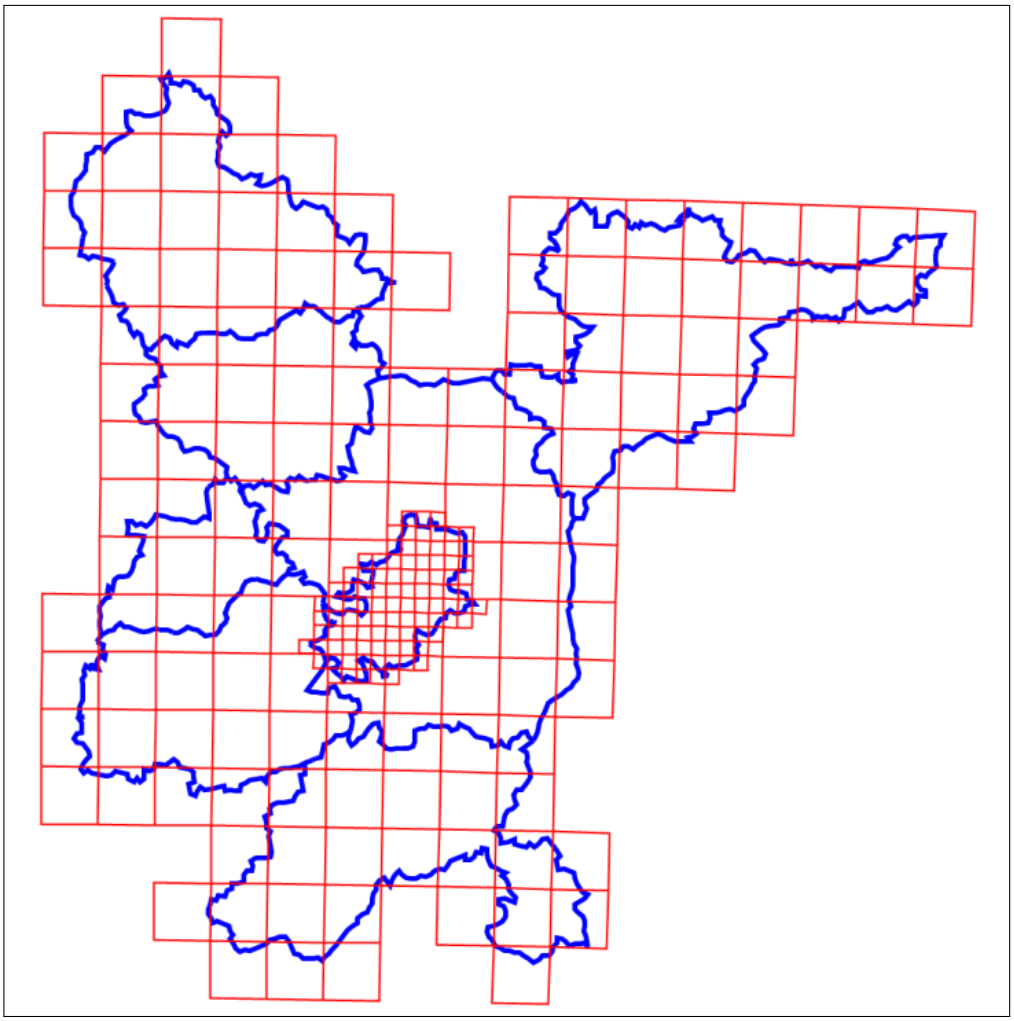


Figura 14 – Zona do condado de Vilnius (ZILIONIENE et al., 2019)

O próximo passo foi, com base na figura 14, obter o modelo da rede rodoviária usando grafos a partir das características das estradas, conforme a figura 15. O grafo possui um total de 5871 km de estradas, 590 nós e 250 centróides. Para o teste em questão, os autores utilizaram apenas as estradas principais, linhas verdes, e nacionais, linhas azuis, que representam 865 km e 1.713 km respectivamente.

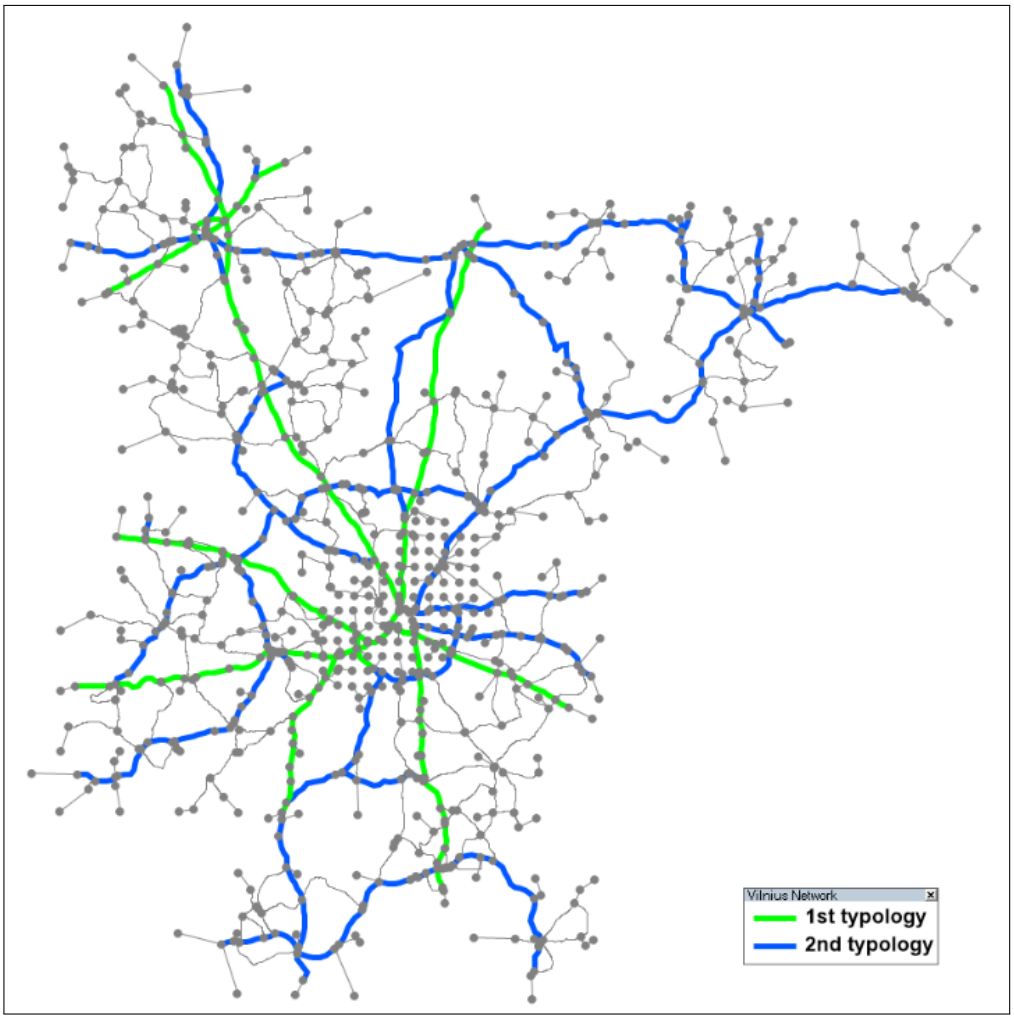


Figura 15 – Modelo de rede condado de Vilnius (ZILIONIENE et al., 2019)

Feita a execução completa do algoritmo, os autores obtiveram os resultados ilustrados na figura 16, onde estão sinalizadas as estradas a serem melhoradas para melhor escoamento da rede rodoviária local.

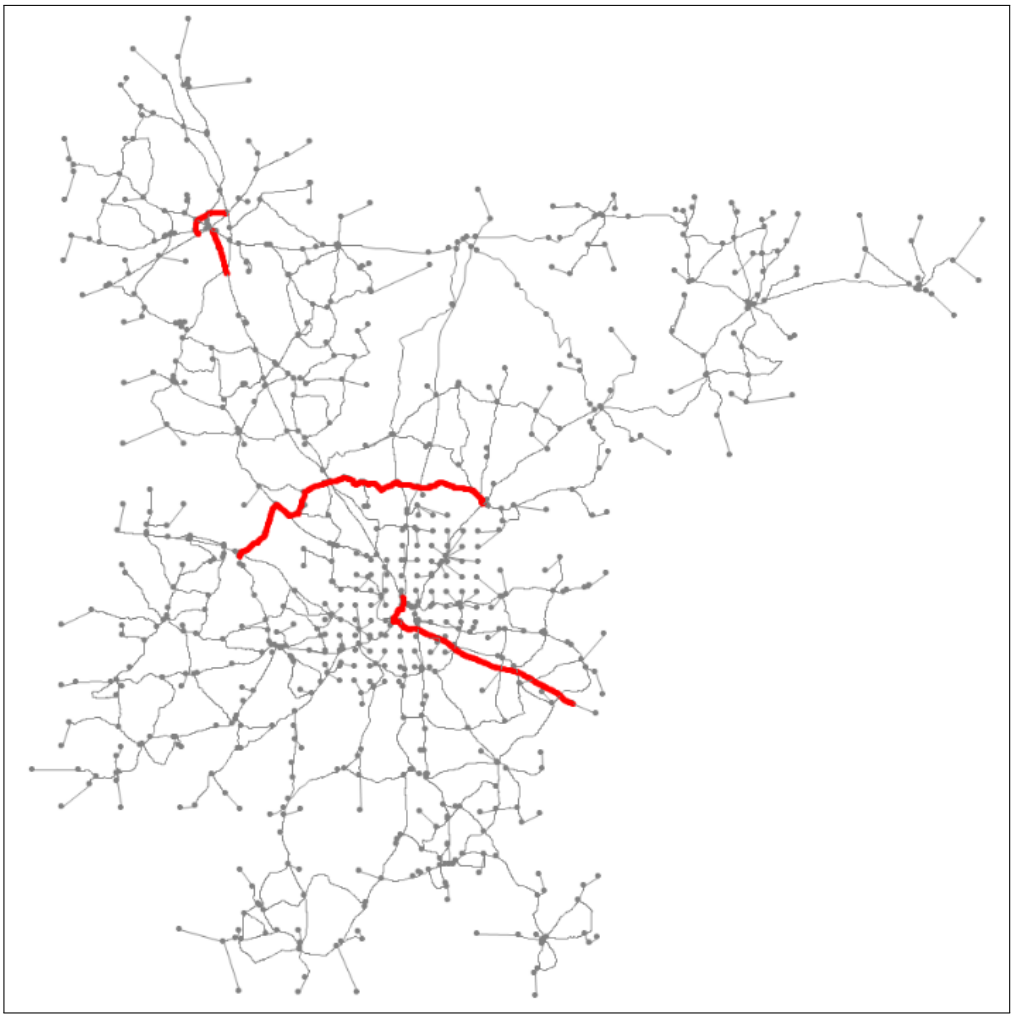


Figura 16 – Estradas a serem melhoradas no condado de Vilnius (ZILIONIENE et al., 2019)

Segundo os autores, os principais resultados da abordagem proposta é a redução do tempo computacional em 40,74% em uma análise anual, e tal valor pode dobrar no caso específico de uma análise do tráfego dos horários matutinos em dias úteis. Obtendo assim respostas mais rápidas e com um custo computacional consideravelmente mais baixo. Para os autores, essa redução só foi possível pelo fato de o algoritmo MSA utilizado adotar os fluxos iniciais de equilíbrio de tráfego baseado na iteração anterior do algoritmo de solução da rede, enquanto na abordagem tradicional, utiliza-se os fluxos iniciais assumindo valor igual a 0.

4 Especificação e Métodos

Neste capítulo, são especificados alguns pontos do trabalho como: apresentação de um algoritmo genético e seus operadores, a representação de um indivíduo, a população inicial, o critério de avaliação do modelo proposto e o plano de experimentação do mesmo.

4.1 Algoritmo genético

O algoritmo genético recebe como dados de entrada um conjunto de estradas e dados relacionados, tais como *travel time* e velocidade máxima de cada via. Através disso, o algoritmo genético inicia sua execução em busca do projeto de trânsito com o melhor *fitness* (inspirado no modelo de Salman e Alaswad (2018), descrito na Seção 3.1).

Para os dados de entrada que serão obtidos a partir da ferramenta OpenStreetMap, as informações capturadas serão: a malha viária da região, com a velocidade máxima de cada via, a quantidade de pistas por via, direção de cada pista. Com esses dados, pretende-se conseguir representar o projeto de trânsito atual da cidade. A ideia é a de que o plano da cidade seja utilizado como base para os indivíduos gerados na população inicial e nas demais etapas do algoritmo genético.

As subseções a seguir apresentam os aspectos considerados durante o projeto do algoritmo genético.

4.1.1 Representação de um indivíduo

Cada indivíduo (cromossomo) representará um projeto de trânsito da cidade, inspirado nas vias existentes da região a ser analisada. Cada segmento de uma via será considerado um gene, ou seja, uma parte do cromossomo. Cada gene possuirá os seguintes atributos: o sentido da via, a quantidade de faixas já existentes e dados de localização. Como o algoritmo genético considerará a possibilidade de criação de novas vias, gene novos podem ser adicionados a um cromossomo. Durante a execução do algoritmo, esses atributos são manipulados pelos operadores genéticos afim de aumentar o *fitness* do modelo.

4.1.2 Operadores genéticos

O modelo proposto neste trabalho utiliza os operadores genéticos descritos na seção 2.2.2, sendo eles: Seleção, Crossover e Mutação.

4.1.2.1 Seleção

Neste trabalho, foram implementadas três técnicas de seleção diferentes: "Torneio", "Roleta" e "Aleatória". A técnica de "Torneio" consistiu em selecionar aleatóriamente 5 dentre a geração atual de indivíduos, após esse passo, é selecionado para a próxima etapa o indivíduo que apresentar o melhor *fitness* entre os 5 selecionados previamente.

A técnica de "Roleta" consistiu em, primeiramente, somar todos os valores de *fitness* de todos os indivíduos. Tendo esse valor em mãos, é escolhido um valor aleatório entre 0 e o total da soma feita anteriormente. Então é realizada uma iteração entre todos os indivíduos da geração atual, incrementando o valor de uma variável com seu valor de *fitness*, e verificando se o valor dessa variável é maior que o valor aleatório escolhido anteriormente. Caso a condição descrita anteriormente seja verdadeira, o indivíduo é escolhido, caso contrário, ignora-se o indivíduo e a iteração é continuada.

Por último, a técnica "Aleatória" escolhe um indivíduo dentre todos os presentes na atual geração.

4.1.2.2 Crossover

O método de *crossover* desenvolvido para este trabalho consiste em analisar cada alteração, em relação ao plano viário original, dos indivíduos pais. Cada alteração dos pais tem 50% de chance de integrar indivíduo filho.

4.1.2.3 Mutação

A técnica de Mutação apresentada nesse trabalho consiste em, com base na taxa de mutação definida no início da execução do algoritmo, modificar aleatóriamente apenas as estradas pré-existentes no plano viário original. As modificações variam entre: inserção de uma nova faixa num trecho de estrada ou inserção de uma nova faixa em todos os trechos de uma estrada.

4.1.3 Fitness

O *fitness* no modelo proposto por este trabalho utiliza, diferentemente do modelo de (SALMAN; ALASWAD, 2018) que utiliza a densidade máxima de veículos numa estrada, o inverso da densidade média de veículos de todas as estradas do plano viário. A fórmula abaixo descreve de maneira detalhada a obtenção deste valor:

$$DM = \frac{DC \cdot 100}{DS},$$

onde DM representa a densidade média, DC representa o número de estradas do plano e DS o somatório das densidades de veículos de todas as estradas. O indivíduo que obtiver o maior resultado ao final deste cálculo será considerado o mais apto.

4.1.4 População inicial

A população inicial a ser utilizada pelo modelo desenvolvido nesse trabalho consiste num conjunto de N indivíduos, ou seja, N possíveis projetos de otimização da malha viária de uma determinada região.

Para definir a população inicial, os N indivíduos deverão conter mudanças aleatoriamente selecionadas a partir do plano atual da região de entrada do algoritmo (bairro, cidade, estado, ...).

4.2 Delineamento do experimento

Para a execução dos experimentos foi utilizado um notebook com as seguintes características:

- Marca: Dell.
- Mémoria RAM: 32 GB.
- Memória: 2TB.
- Processador: Intel core i7 11th.
- Placa de vídeo: NVIDIA GP108M [GeForce MX330].

Com as características acima descritas, este notebook levou cerca de duas horas para executar as 72 combinações de conjuntos de parâmetros possíveis, sendo repetida 10 vezes para cada conjunto.

4.3 Roteiro de experimentos

Os experimentos efetuados neste trabalho são basicamente execuções do modelo com variações dos parâmetros existentes no algoritmo genético, com objetivo de detectar em quais gerações acontece convergência entre eles, considerando o tamanho da instância de entrada (número de trechos da via). Com isso, fica evidente a influência de cada parâmetro sobre o resultado da execução do modelo.

Para a execução dos testes, foi selecionado como plano viário base o município de Borá, localizado no interior do estado de São Paulo e sendo considerado o menor município do estado segundo a revista Veja (2019) no quesito população. A escolha por uma cidade pequena se deu pela maior praticidade na execução dos experimentos de acordo com os recursos computacionais presentes no momento. Outros dois parâmetros foram deixados de maneira fixa durante todo o experimento, são eles: Número de gerações(50) e *budget* disponível para alteração na malha viária, sendo este de 10 quilômetros. A lista abaixo,

ilustra os demais parâmetros que serão alterados durante a execução do algoritmo genético e seus valores, respectivamente:

- População inicial: 10, 15 e 20 Indivíduos.
- Taxa de mutação: 1% e 10%.
- Taxa de indivíduos elitistas por geração: 10% e 20%.
- Técnicas de seleção: Torneio, Roleta e Aleatória.
- Taxa de filhos gerados por geração: 25% e 50%.

Tomando tais parâmetros como base, será executado o algoritmo genético 10 vezes com cada uma das 72 combinações existentes.

4.4 Análise de Resultados: o que foi analisado

Feita a execução de todos os cenários descritos anteriormente em 4.3, são efetuadas as seguintes análises:

- Através de gráficos, definir uma tendência de comportamento do algoritmo, apresentando os conjuntos de parâmetros que obtiveram, em média, os melhores indivíduos.
- Através de gráficos e de uma tabela, ilustrar os indivíduos que obtiveram os melhores *fitness*, bem como as modificações efetuadas pelo algoritmo em cada um deles.
- Definição do melhor indivíduo encontrado dentre todas as execuções do algoritmo genético. Sendo este, o indivíduo que possuir o maior *fitness*.

4.5 Resultados

Esta seção se destina a exibir e comentar os resultados obtidos.

4.5.1 Melhores conjuntos de parâmetros

Com base nos dados obtidos, foi efetuado um cálculo de média, onde soma-se os valores de média de *fitness* por geração em cada execução, conforme ilustrado abaixo:

$$FMCP = \frac{\frac{SMFG_1}{NE} + \frac{SMFG_2}{NE} + ... + \frac{SMFG_G}{NE}}{G} \tag{4.1}$$

no qual, $FMCP$ indica o *fitness* médio por conjunto de parâmetros. O $SMFG_i$ representa o somatório das médias de *fitness* da i-ésima geração em todas as execuções do algoritmo

com determinado conjunto de parâmetros. NE representa o número de execuções do algoritmo, já G representa o número de gerações pré-definidas. Essa equação é utilizada nas Tabelas 4.5.2 e 6.1.

Dado este cenário, obteve-se como resultado as 10 melhores médias de *fitness* por conjunto de parâmetros, conforme gráfico da Figura 17. Para o gráfico, é considerado "fitness" a equação abaixo:

$$\text{Fitness por geração} = \frac{SMFG_i}{NE}. \tag{4.2}$$

no qual i corresponde a i-ésima geração.

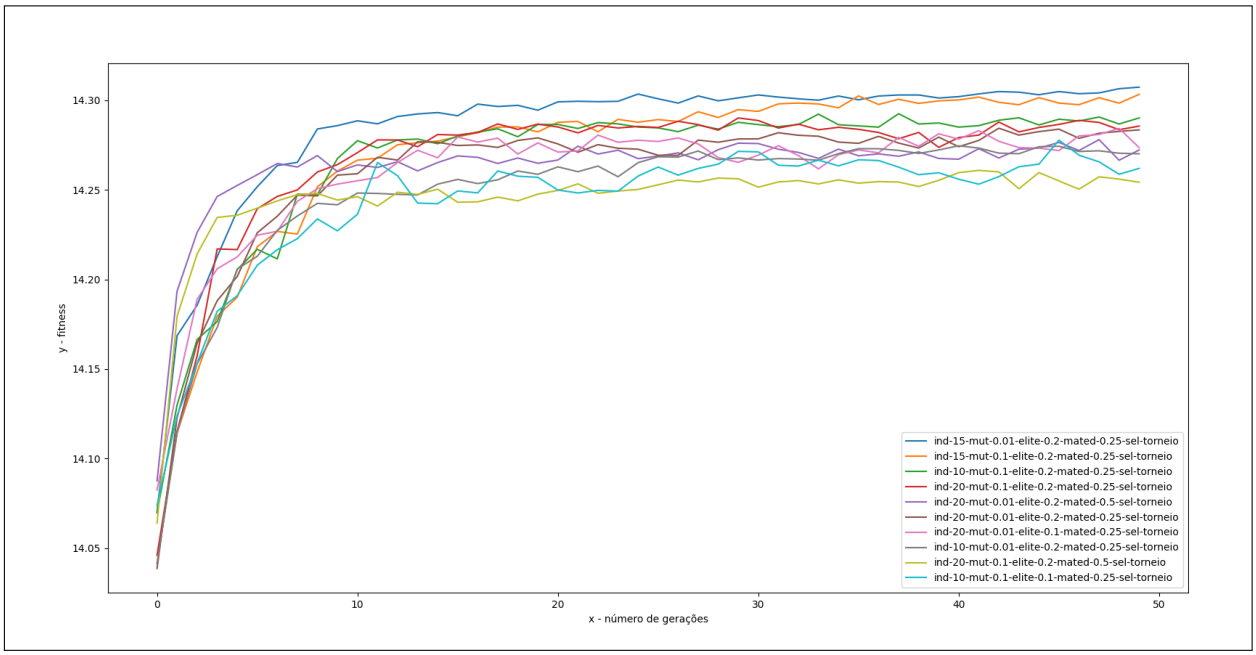


Figura 17 – Os 10 melhores conjuntos de parâmetros com base no *fitness* médio por geração.

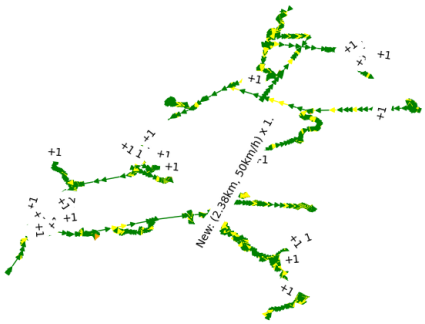
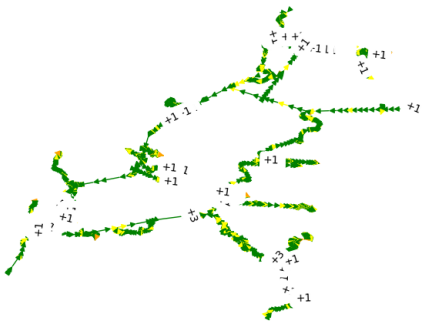
Analisando o gráfico da Figura 17, pode-se concluir que o algoritmo tende a apresentar os melhores indivíduos, em média, quando a técnica de seleção utilizada é a de "Torneio", visto que todos os registros presentes no gráfico utilizaram tal técnica. Há outros dois pontos a serem observados, o primeiro refere-se as taxas de indivíduos elitistas e de filhos gerados através da etapa de *crossover*. Neste primeiro ponto, percebe-se que tais parâmetros são importantes na obtenção do resultados, mas de maneiras opostas. A taxa de elitismo se comportou de maneira progressiva, ou seja, quanto maior, melhor o resultado. Com relação ao parâmetro taxa de filhos gerados através da etapa de *crossover*, observa-se que quanto maior o número de filhos, menor a qualidade esperada por geração. Já o segundo ponto, refere-se a taxa de mutação e o tamanho da população, tais fatores não se mostraram relevantes, pois seus resultados são equilibrados. Portanto, após os experimentos verificou-se que, em 50 gerações e com 10 execuções do algoritmo

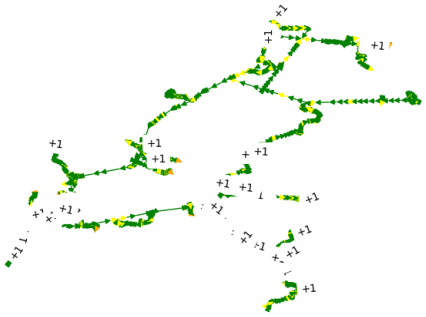
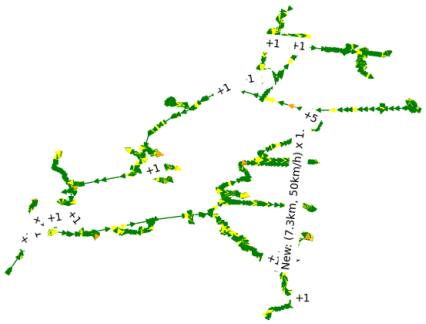
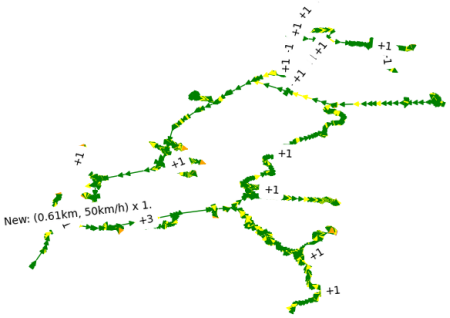
com cada conjunto de parâmetros, os melhores resultados obtidos possuíam os seguintes parâmetros:

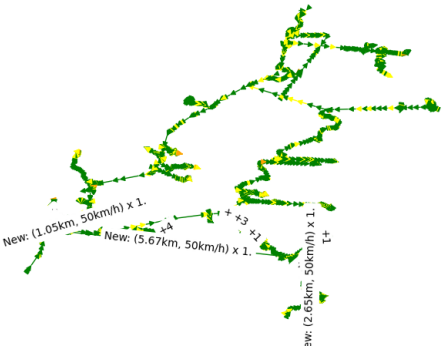
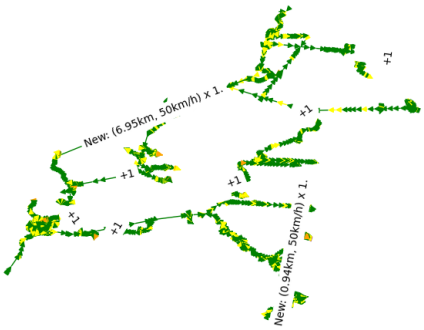
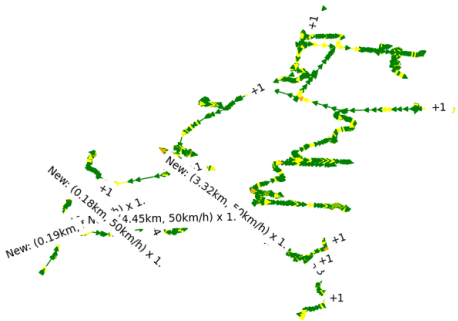
- População Inicial: 15 indivíduos.
- Taxa de mutação: 1%.
- Taxa de indivíduos elitistas por geração: 20%.
- Taxa de filhos gerados por geração: 25%.
- Técnica de seleção: Torneio.

4.5.2 Melhores indivíduos encontrados por conjunto de parâmetros

Para encontrar os melhores indivíduos por conjunto de parâmetros individualmente, foi feita uma análise pelo *fitness* do melhor indivíduo de cada execução. Dado esse cenário, gerou-se uma classificação com os melhores indivíduos, onde também é ilustrado o layout do plano de trânsito de cada um dos classificados.

Colocação	**Conj. parâmetros**	**Fitness**	**Layout**
1º	Indivíduos - 15 Tx. Mutação - 10% Tx. Elitismo - 20% Tx. Filhos gerados - 50% Téc. Seleção - Torneio	14.4661	
2º	Indivíduos - 20 Tx. Mutação - 10% Tx. Elitismo - 20% Tx. Filhos gerados - 50% Téc. Seleção - Torneio	14.4102	

Colocação	**Conj. parâmetros**	**Fitness**	**Layout**
3º	Indivíduos - 15 Tx. Mutação - 10% Tx. Elitismo - 10% Tx. Filhos gerados - 25% Téc. Seleção - Torneio	14.4015	
4º	Indivíduos - 10 Tx. Mutação - 10% Tx. Elitismo - 10% Tx. Filhos gerados - 50% Téc. Seleção - Torneio	14.4005	
5º	Indivíduos - 15 Tx. Mutação - 1% Tx. Elitismo - 10% Tx. Filhos gerados - 50% Téc. Seleção - Torneio	14.3926	

Colocação	Conj. parâmetros	Fitness	Layout
6º	Indivíduos - 20 Tx. Mutação - 10% Tx. Elitismo - 10% Tx. Filhos gerados - 25% Téc. Seleção - Torneio	14.3863	
7º	Indivíduos - 15 Tx. Mutação - 1% Tx. Elitismo - 20% Tx. Filhos gerados - 50% Téc. Seleção - Torneio	14.3862	
8º	Indivíduos - 20 Tx. Mutação - 10% Tx. Elitismo - 20% Tx. Filhos gerados - 25% Téc. Seleção - Torneio	14.3794	

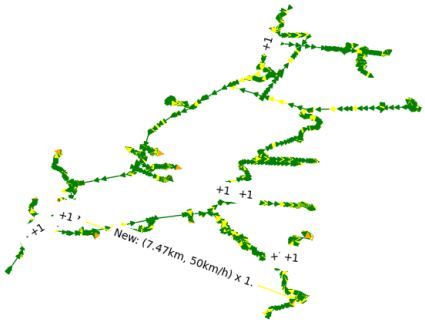
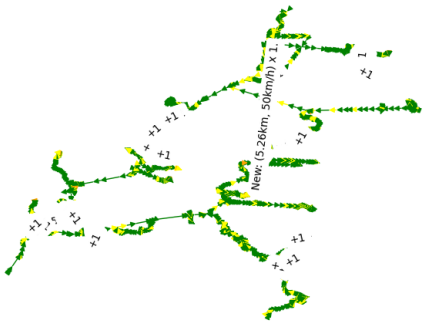
Colocação	**Conj. parâmetros**	**Fitness**	**Layout**
9º	Indivíduos - 20 Tx. Mutação - 1% Tx. Elitismo - 10% Tx. Filhos gerados - 25% Téc. Seleção - Torneio	14.3745	
10º	Indivíduos - 15 Tx. Mutação - 10% Tx. Elitismo - 10% Tx. Filhos gerados - 50% Téc. Seleção - Torneio	14.3745	

Tabela 1 – Os 10 melhores indivíduos encontrados por conjunto de parâmetros

Ao observarmos a tabela 4.5.2, é possível verificar as diferentes alterações efetuadas pelo algoritmo genético no plano viário da cidade de Borá-SP. As indicações que contenham um "+" acompanhado de um determinado valor N, indicam que houve um acréscimo de N faixas de trânsito naquele trecho de via. Já a outra indicação, sugere que houve a inserção de uma nova estrada no plano viário preexistente, de tamanho, velocidade máxima permitida e número de faixas também explicitados. As cores nos mapas indicam a intensidade do transito nas vias, quanto mais quente for a cor, mais intenso o trânsito. A Figura 18 mostra a evolução do *fitness* dos indivíduos classificados no decorrer das 50 gerações.

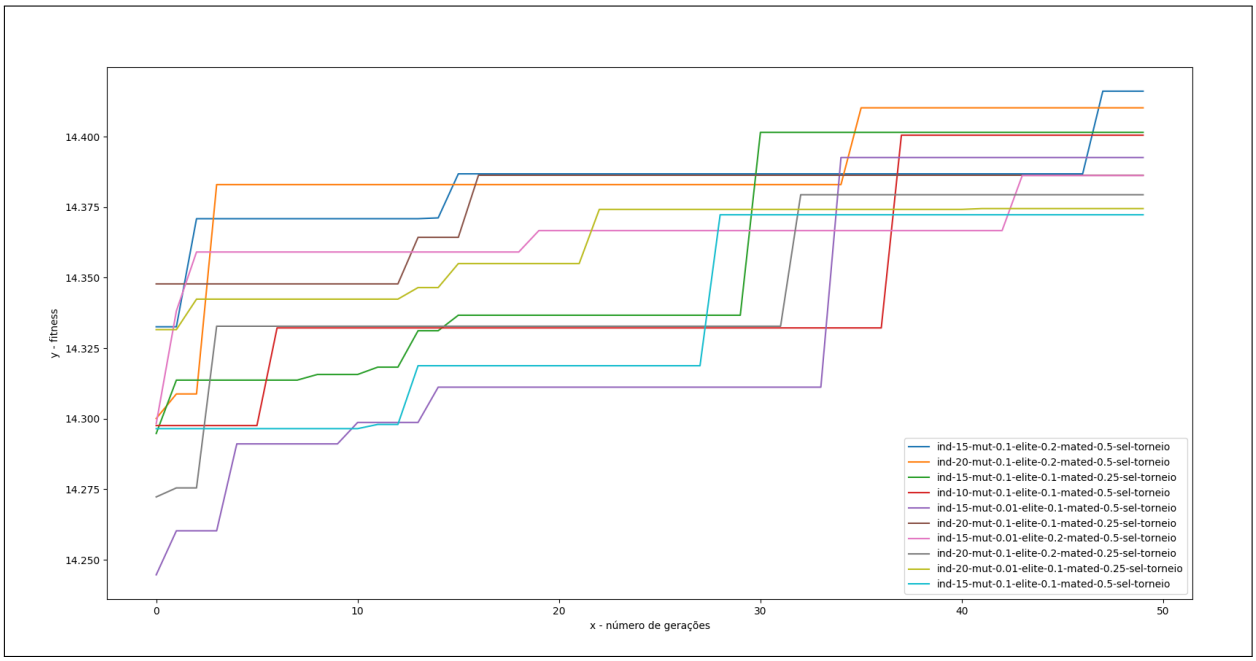


Figura 18 – Os 10 melhores indivíduos com base no *fitness* por conjunto de parâmetros.

Com base nas informações do gráfico da Figura 18, percebe-se que novamente a técnica de seleção que apresentou os melhores resultados, assim como na seção 4.5.1, é a de "Torneiro". Portanto, pode-se concluir que o parâmetro mais influente nos resultados foi a técnica de seleção empregada, já os demais parâmetros demonstram-se equilibrados durante os experimentos. Outro ponto a ser destacado é o fato de nenhum dos registros do gráfico apresentar declínio durante o decorrer das gerações, isso se deu pelo fato do algoritmo possuir um caráter elitista. Isso garante que ao final da execução, teremos sempre o melhor indivíduo dentre todos os gerados.

4.5.3 Melhor indivíduo encontrado

Após a execução dos experimentos com base no plano viário apresentado na Figura 19, o indivíduo que apresentou o melhor *fitness* com base nos parâmetros, fixos e variáveis, foi o da Figura 20 ilustrada abaixo

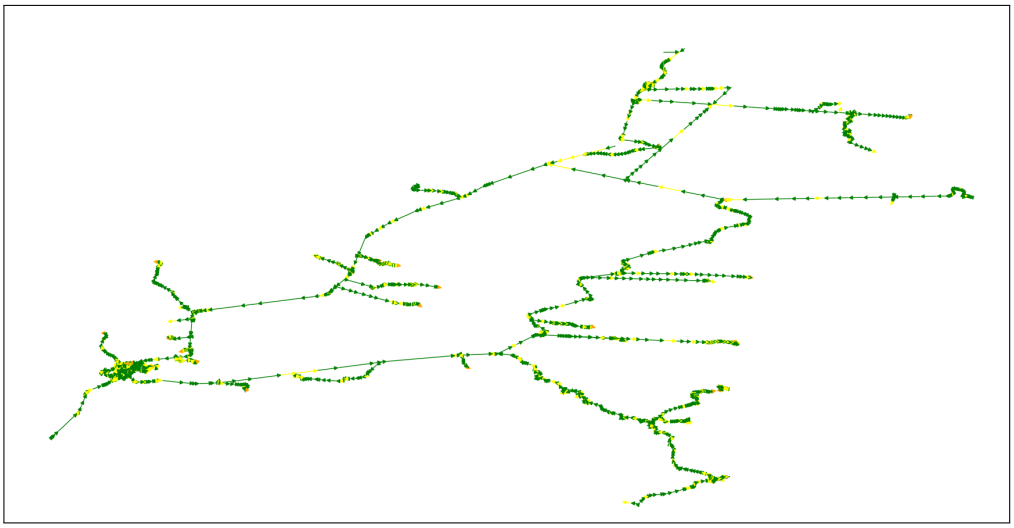


Figura 19

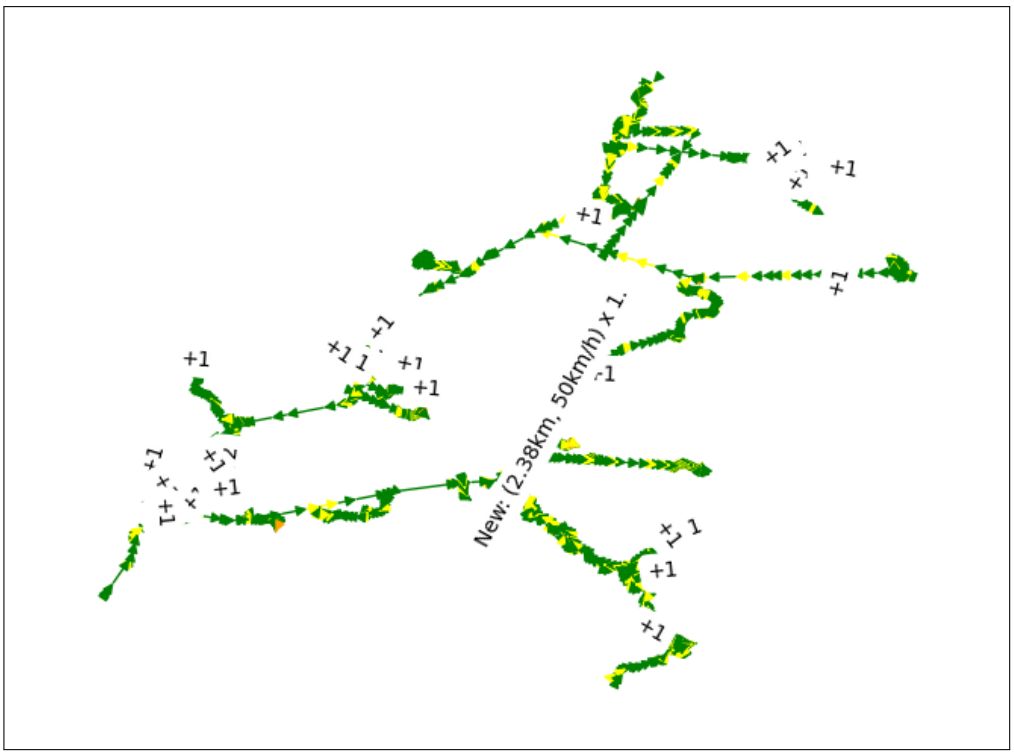


Figura 20 – O melhor indivíduo encontrado durante os experimentos.

Comparado com o plano viário inicial do experimento, o indivíduo selecionado apresentou cerca de 27 faixas novas em trechos de estradas preexistentes e a inserção de uma nova pista de 2,38KM no plano viário da cidade analisada. Além disso, possui um *fitness* no valor de 14.4161, sendo levemente superior aos demais indivíduos. O conjunto de parâmetros utilizado para encontrar este indivíduo possui:

- Budget: 10 quilômetros.

- Número de gerações: 50 gerações.
- População Inicial: 15 indivíduos.
- Taxa de mutação: 10%.
- Taxa de indivíduos elitistas por geração: 20%.
- Taxa de filhos gerados por geração: 50%.
- Técnica de seleção: Torneio.

5 Conclusões

Com o objetivo de otimizar a malha viária de uma determidada localidade reduzindo o número de congestionamentos, a abordagem trazida por este trabalho contempla a utilização de conceitos de computação evolutiva, mais especificamente a implementação de um algoritmo genético para tal tarefa.

Este trabalho apresenta um algoritmo genético capaz de modificar o plano viário de uma localidade, seja um bairro ou uma cidade, de forma a melhorar o trânsito. Para atingir esse objetivo, foi necessário efetuar um levantamento de trabalhos relacionados ao tema na literatura acadêmica, definir de uma localidade a ser analisada e o desenvolver um algoritmo genético adequado.

Após a realização dos experimentos descritos em 4.5, encontrou-se:

- o melhor plano viário, encontrado durante os experimentos, para a cidade de Borá-SP sendo o indivíduo representado na Figura 20, que possui um *fitness* de 14.4661.
- o conjunto de parâmetros que apresentou os melhores resultados, descrito em 4.5.1.
- o melhor indivíduo de cada combinação de parâmetros encontrado durante os experimentos.

Todos os objetivos definidos para este trabalho foram atingidos, porém sabe-se que as soluções encontradas não são as melhores possíveis, mas sim as melhores encontradas dentros dos parâmetros existentes e dos experimentos efetuados.

Trabalhos futuros

Abaixo são apresentados possíveis temas para trabalhos futuros:

- adaptar o algoritmo para levar em consideração a análise de *travel time* (tempo de viagem) e não mais a densidade média de veículos;
- expandir o número de cidades analisadas;
- incorporar os recursos financeiros ao budget, para contemplar não apenas a quilometragem disponível;
- incorporar a análise de relevo ao algoritmo, para que não haja inserção de vias onde financeiramente não é possível.

Referências Bibliográficas

BARCELLOS, J. C. H. de. Algoritmos genéticos adaptativos: um estudo comparativo. 2000. 17

CAGGIANI, L.; CAMPOREALE, R.; OTTOMANELLI, M. Facing equity in transportation network design problem: A flexible constraints based model. **Transport Policy**, v. 55, p. 9–17, 01 2017. 21, 22

CARVALHO, A. 2009. Disponível em: <https://sites.icmc.usp.br/andre/research/genetic/>. 8, 18

DARWIN, C. R. **A origem das espécies**. 1st ed.. ed. [S.l.]: Edipro, 1859. ISBN 8552100150. 16

DETRAN, D. estadual de trânsito. **Veículos em circulação em Santa Catarina**. 2021. Disponível em: <https://www.detran.sc.gov.br/estatisticas/veiculos>. 12

DUELL, M.; GARDNER, L.; WALLER, S. Policy implications of incorporating distance constrained electric vehicles into the traffic network design problem. **Transportation Letters**, v. 10, p. 1–15, 10 2016. 21

GALLO, M.; D'ACIERNO, L.; MONTELLA, B. A meta-heuristic approach for solving the urban network design problem. **European Journal of Operational Research**, v. 201, p. 144–157, 02 2010. 21

GESTAL, M. Introduccion a los algoritmos geneticos. 08 2013. 17, 18, 20

GOLDBERG, D. E. **Genetic Algorithms in Search, Opti-mization and Machine Learning.** 13th ed.. ed. [S.l.]: Addison-Wesley Professional, 1989. ISBN 0201157675. 17

GRIGOLETTI, P. Cadeias de markov. 08 2015. 8, 15

IBGE, I. brasileiro de geografia e estatística. **Estatísticas gerais**. 2021. Disponível em: <https://www.ibge.gov.br/cidades-e-estados/sc.html>. 12

IT-BRAIN. **Algoritmos genéticos - Guia rápido**. 2020. Disponível em: <https://pt.it-brain.online/tutorial/genetic_algorithms/genetic_algorithms_quick_guide/>. 8, 19, 20

LACERDA, E.; CARVALHO, A. de; LUDERMIR, T. Um tutorial sobre algoritmos genéticos. **RITA**, v. 9, p. 7–39, 01 2002. 17, 20

LEBLANC, L. An algorithm for the discrete network design problem. **Transportation Science**, v. 9, p. 183–199, 08 1975. 12

LOPES, H. S.; TAKAHASHI, R. H. C. **Computação evolucionária em problemas de engenharia**. [S.l.]: Omnipax, 2011. 16

MACIEL, V. Congestionamentos urbanos. **GV-executivo**, v. 7, p. 20, 10 2008. 12

POORZAHEDY, H.; ROUHANI, O. M. Hybrid meta-heuristic algorithms for solving network design problem. **European Journal of Operational Research**, v. 182, p. 578–596, 02 2007. 22

POORZAHEDY, H.; TURNQUIST, M. Approximate algorithms for the discrete network design problem. **Transportation Research Part B: Methodological**, v. 16, p. 45–55, 02 1982. 12

SALMAN, S.; ALASWAD, S. Alleviating road network congestion: Traffic pattern optimization using markov chain traffic assignment. **Computers & Operations Research**, v. 99, 07 2018. 8, 12, 13, 14, 15, 21, 24, 25, 32, 33

SILVA, L. M. D. Cadeias de markov e aplicações. 2017. Disponível em: <https://app.uff.br/riuff/handle/1/4213>. 16

TOMASZEWSKI, M.; SANTIAGO, R. de. **Método Computacional para a Otimização do Projeto da Malha Viária de Florianópolis-SC**. 2020. Disponível em: <https://repositorio.ufsc.br/handle/123456789/212551>. 25, 28

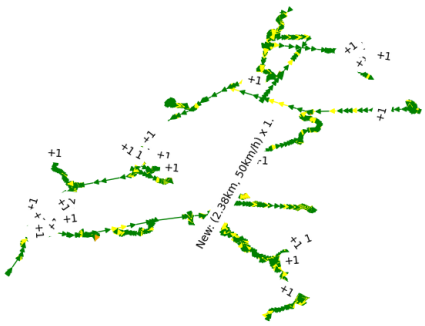
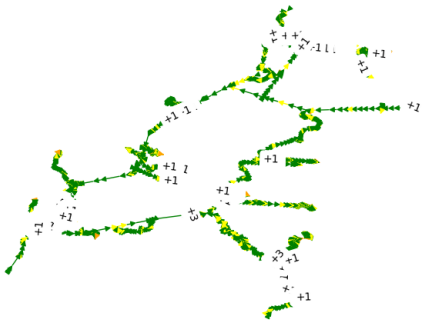
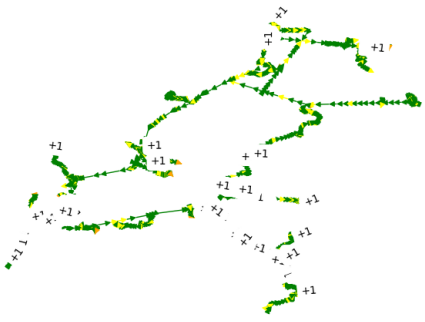
VEJA. **Menor município do estado, Borá teme junção com cidade maior.** 2019. Disponível em: <https://vejasp.abril.com.br/cidades/bora-menor-municipio/>. 34

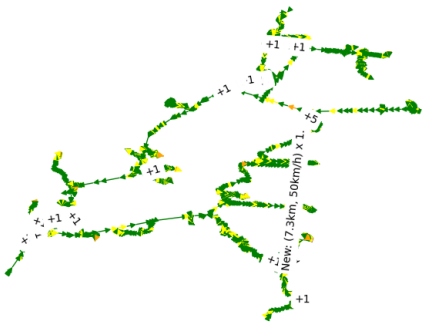
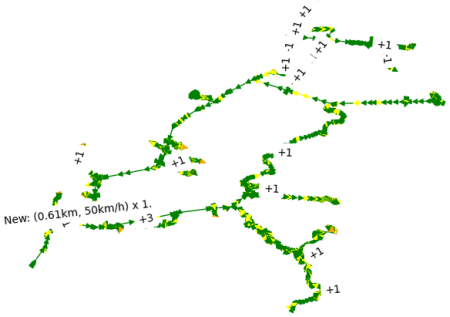
VIANNA, G.; YOUNG, C. E. Em busca do tempo perdido: Uma estimativa do produto perdido em trÂnsito no brasil. **Revista de Economia Contemporânea**, v. 19, p. 403–416, 12 2015. 6, 7

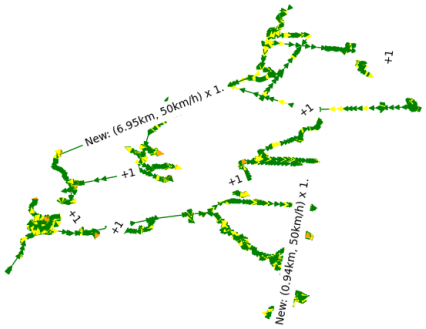
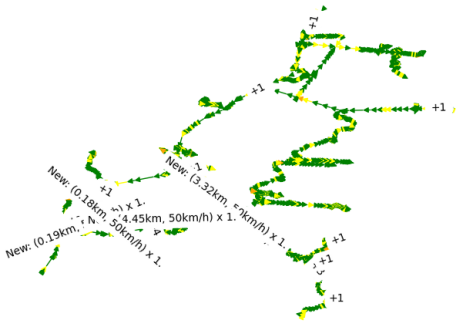
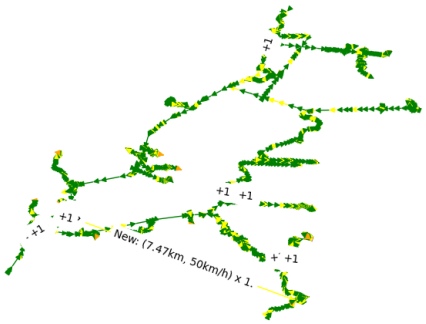
ZILIONIENE, D. et al. A general methodology for reducing computing times of road network design algorithms. **International Journal of Supply and Operations Management**, v. 6, p. 126–141, 05 2019. 8, 29, 30, 31

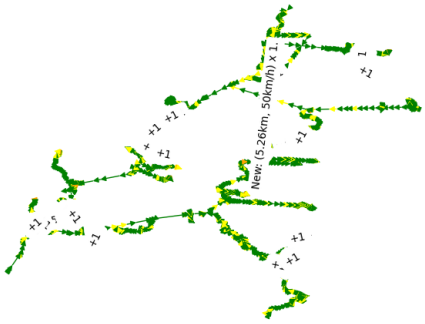
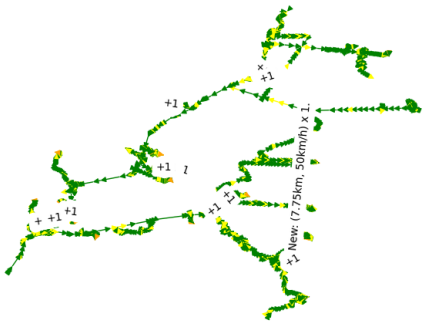
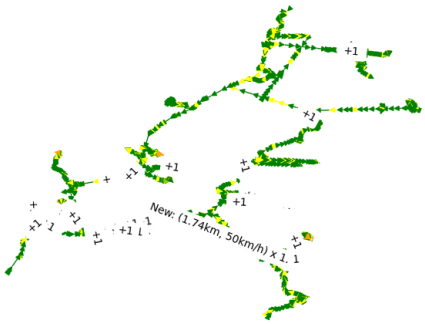
6 Apêndices

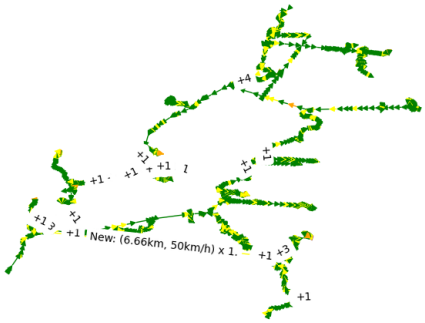
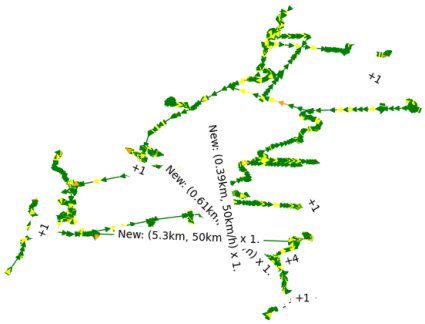
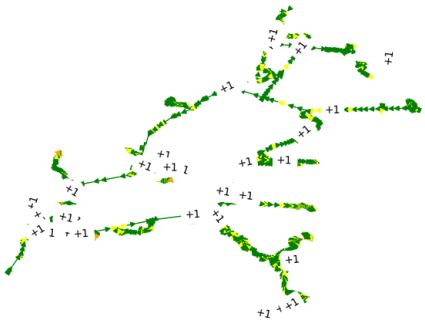
6.1 Classificação com os melhores individuos de todos os conjuntos de parâmetros

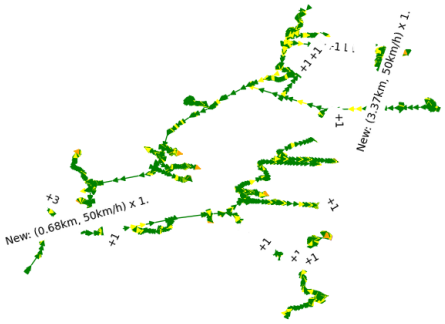
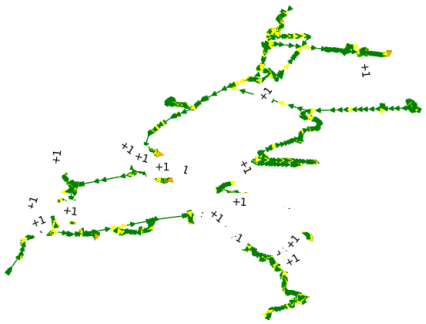
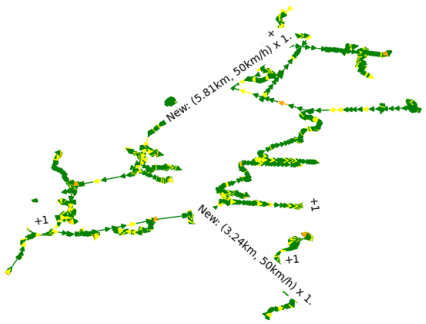
Colocação	Conj. parâmetros	Fitness	Layout
1º	Indivíduos - 15 Tx. Mutação - 10% Tx. Elitismo - 20% Tx. Filhos gerados - 50% Téc. Seleção - Torneio	14.4661	
2º	Indivíduos - 20 Tx. Mutação - 10% Tx. Elitismo - 20% Tx. Filhos gerados - 50% Téc. Seleção - Torneio	14.4102	
3º	Indivíduos - 15 Tx. Mutação - 10% Tx. Elitismo - 10% Tx. Filhos gerados - 25% Téc. Seleção - Torneio	14.4015	

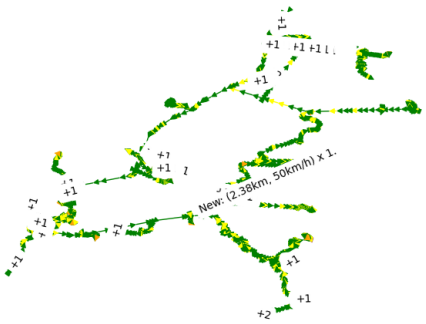
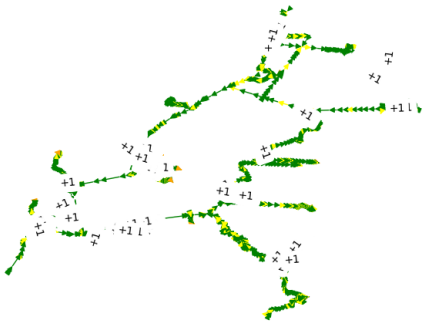
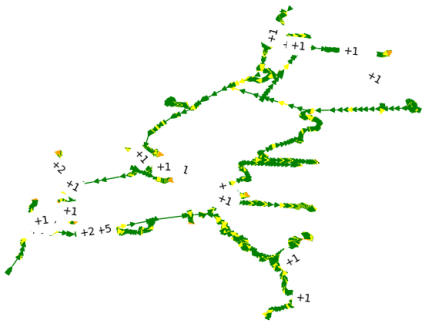
Colocação	**Conj. parâmetros**	**Fitness**	**Layout**
4º	Indivíduos - 10 Tx. Mutação - 10% Tx. Elitismo - 10% Tx. Filhos gerados - 50% Téc. Seleção - Torneio	14.4005	
5º	Indivíduos - 15 Tx. Mutação - 1% Tx. Elitismo - 10% Tx. Filhos gerados - 50% Téc. Seleção - Torneio	14.3926	
6º	Indivíduos - 20 Tx. Mutação - 10% Tx. Elitismo - 10% Tx. Filhos gerados - 25% Téc. Seleção - Torneio	14.3863	

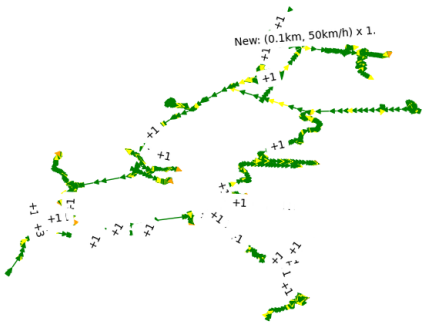
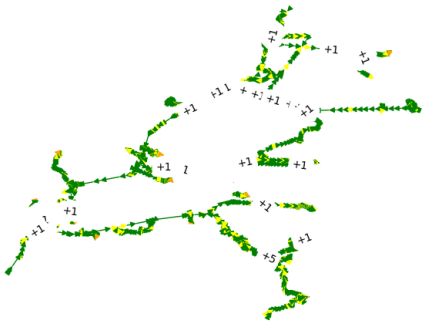
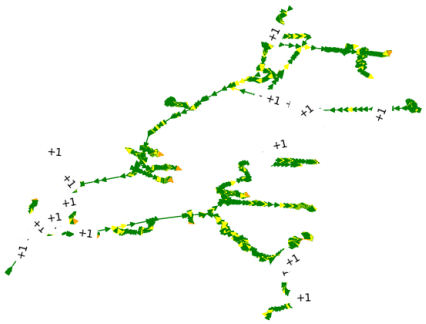
Colocação	Conj. parâmetros	Fitness	Layout
7º	Indivíduos - 15 Tx. Mutação - 1% Tx. Elitismo - 20% Tx. Filhos gerados - 50% Téc. Seleção - Torneio	14.3862	
8º	Indivíduos - 20 Tx. Mutação - 10% Tx. Elitismo - 20% Tx. Filhos gerados - 25% Téc. Seleção - Torneio	14.3794	
9º	Indivíduos - 20 Tx. Mutação - 1% Tx. Elitismo - 10% Tx. Filhos gerados - 25% Téc. Seleção - Torneio	14.3745	

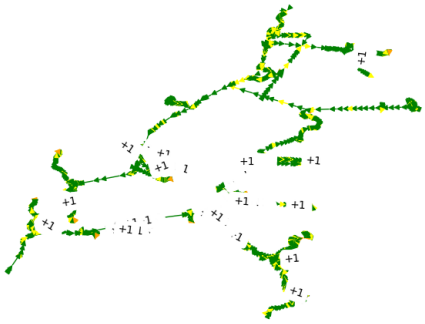
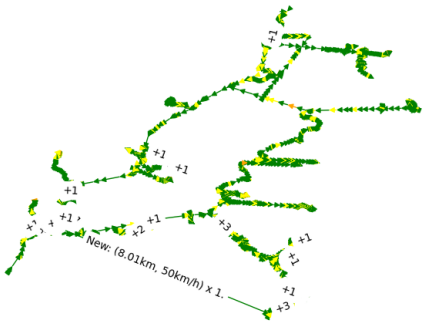
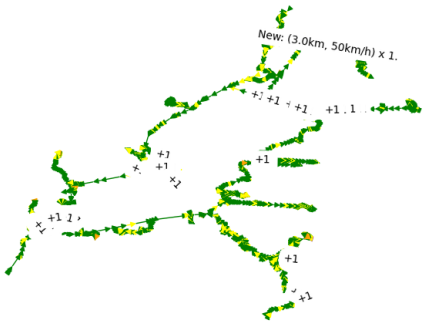
Colocação	Conj. parâmetros	Fitness	Layout
10º	Indivíduos - 15 Tx. Mutação - 10% Tx. Elitismo - 10% Tx. Filhos gerados - 50% Téc. Seleção - Torneio	14.3745	
11º	Indivíduos - 20 Tx. Mutação - 1% Tx. Elitismo - 20% Tx. Filhos gerados - 50% Téc. Seleção - Torneio	14.3713	
12º	Indivíduos - 20 Tx. Mutação - 10% Tx. Elitismo - 10% Tx. Filhos gerados - 50% Téc. Seleção - Torneio	14.3710	

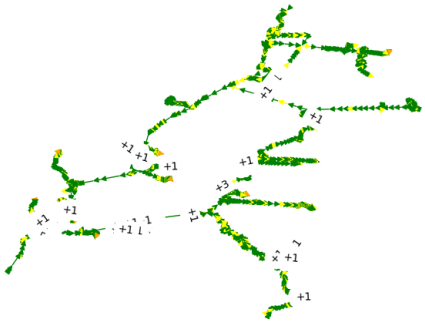
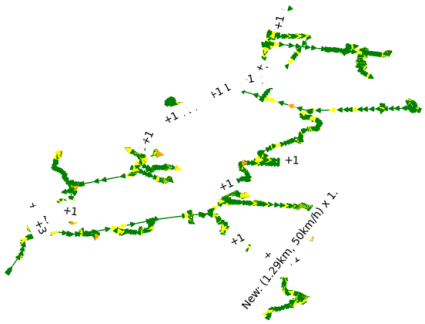
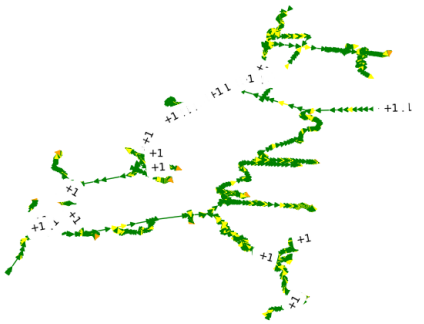
Colocação	**Conj. parâmetros**	**Fitness**	**Layout**
13º	Indivíduos - 15 Tx. Mutação - 1% Tx. Elitismo - 1% Tx. Filhos gerados - 25% Téc. Seleção - Torneio	14.3705	
14º	Indivíduos - 10 Tx. Mutação - 1% Tx. Elitismo - 1% Tx. Filhos gerados - 25% Téc. Seleção - Torneio	14.3692	
15º	Indivíduos - 15 Tx. Mutação - 10% Tx. Elitismo - 20% Tx. Filhos gerados - 50% Téc. Seleção - Aleatória	14.3670	

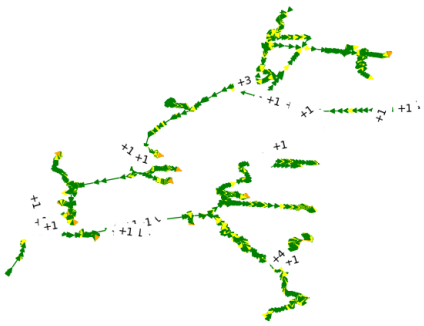
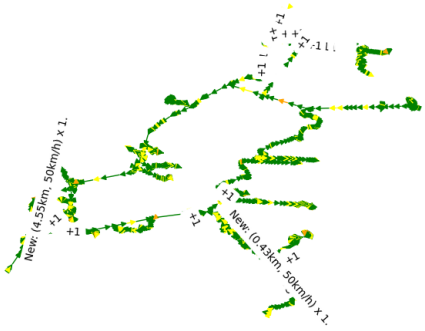
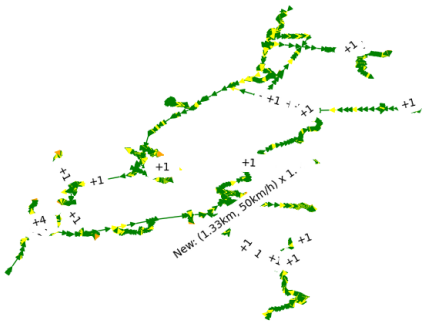
Colocação	Conj. parâmetros	Fitness	Layout
16º	Indivíduos - 20 Tx. Mutação - 1% Tx. Elitismo - 20% Tx. Filhos gerados - 50% Téc. Seleção - Roleta	14.3654	
17º	Indivíduos - 15 Tx. Mutação - 1% Tx. Elitismo - 10% Tx. Filhos gerados - 50% Téc. Seleção - Roleta	14.3651	
18º	Indivíduos - 15 Tx. Mutação - 1% Tx. Elitismo - 20% Tx. Filhos gerados - 50% Téc. Seleção - Aleatória	14.3649	

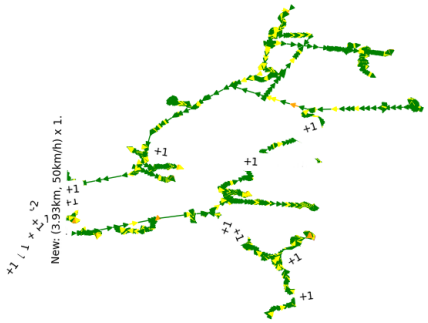
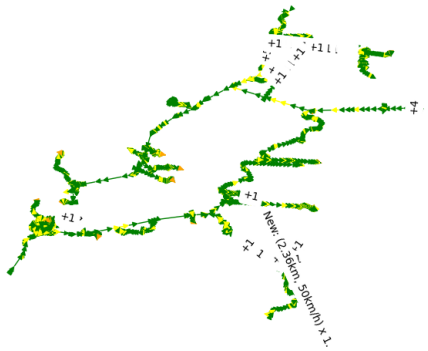
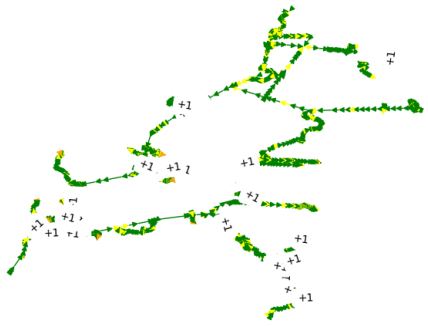
Colocação	Conj. parâmetros	Fitness	Layout
19º	Indivíduos - 20 Tx. Mutação - 1% Tx. Elitismo - 10% Tx. Filhos gerados - 50% Téc. Seleção - Torneio	14.3643	
20º	Indivíduos - 20 Tx. Mutação - 10% Tx. Elitismo - 20% Tx. Filhos gerados - 50% Téc. Seleção - Roleta	14.3641	
21º	Indivíduos - 20 Tx. Mutação - 1% Tx. Elitismo - 10% Tx. Filhos gerados - 50% Téc. Seleção - Aleatória	14.3640	

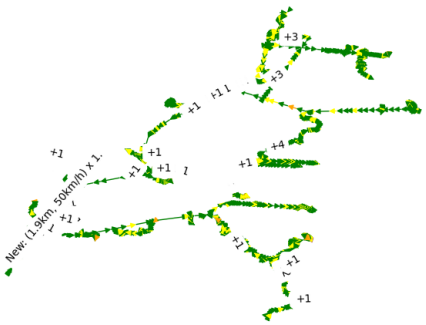
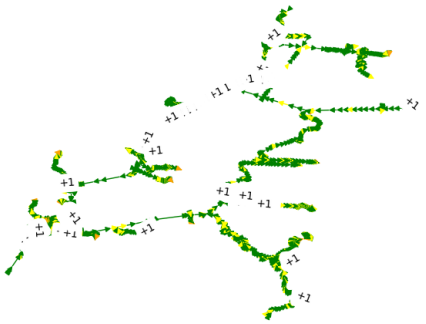
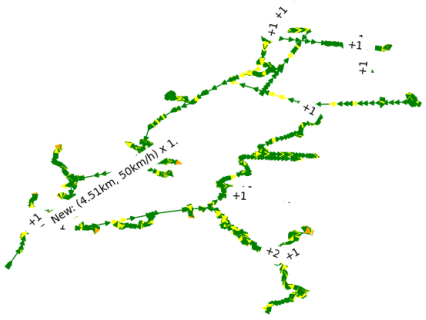
Colocação	Conj. parâmetros	Fitness	Layout
22º	Indivíduos - 20 Tx. Mutação - 10% Tx. Elitismo - 10% Tx. Filhos gerados - 25% Téc. Seleção - Roleta	14.3635	
23º	Indivíduos - 15 Tx. Mutação - 1% Tx. Elitismo - 20% Tx. Filhos gerados - 25% Téc. Seleção - Roleta	14.3624	
24º	Indivíduos - 15 Tx. Mutação - 10% Tx. Elitismo - 20% Tx. Filhos gerados - 25% Téc. Seleção - Aleatória	14.3595	

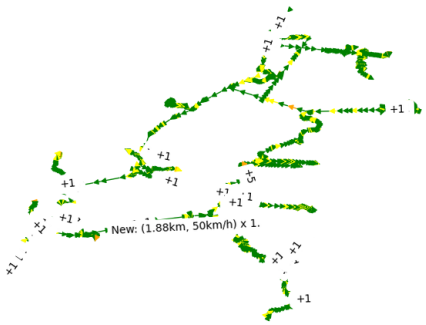
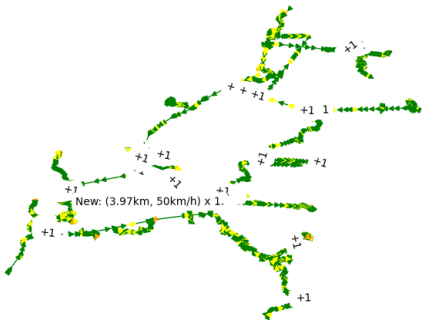
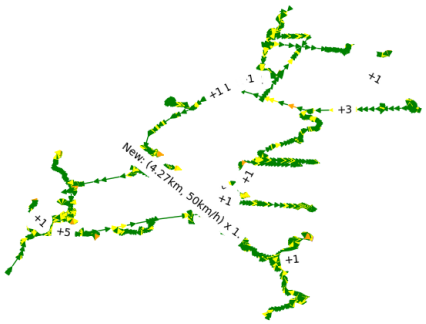
Colocação	Conj. parâmetros	Fitness	Layout
25º	Indivíduos - 15 Tx. Mutação - 10% Tx. Elitismo - 20% Tx. Filhos gerados - 25% Téc. Seleção - Torneio	14.3591	
26º	Indivíduos - 10 Tx. Mutação - 10% Tx. Elitismo - 20% Tx. Filhos gerados - 50% Téc. Seleção - Torneio	14.3589	
27º	Indivíduos - 15 Tx. Mutação - 1% Tx. Elitismo - 10% Tx. Filhos gerados - 25% Téc. Seleção - Roleta	14.3584	

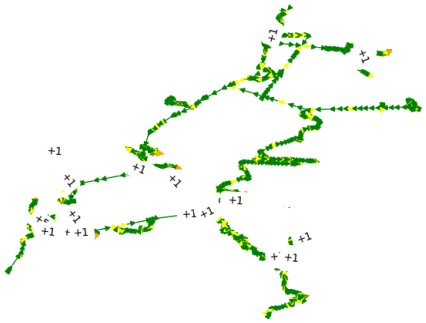
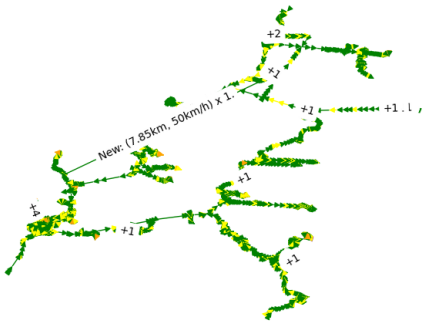
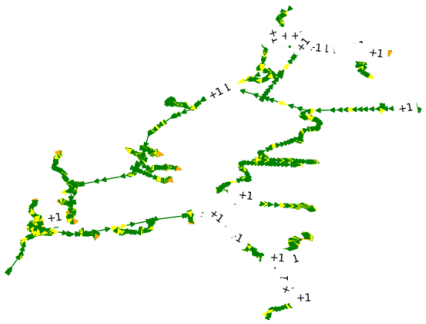
Colocação	**Conj. parâmetros**	**Fitness**	**Layout**
28º	Indivíduos - 20 Tx. Mutação - 1% Tx. Elitismo - 20% Tx. Filhos gerados - 50% Téc. Seleção - Aleatória	14.3573	
29º	Indivíduos - 20 Tx. Mutação - 1% Tx. Elitismo - 20% Tx. Filhos gerados - 25% Téc. Seleção - Aleatória	14.3572	
30º	Indivíduos - 20 Tx. Mutação - 1% Tx. Elitismo - 20% Tx. Filhos gerados - 25% Téc. Seleção - Torneio	14.3568	

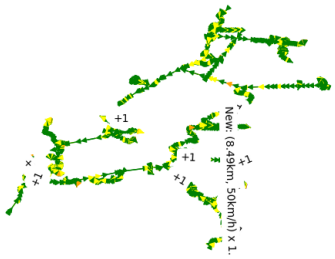
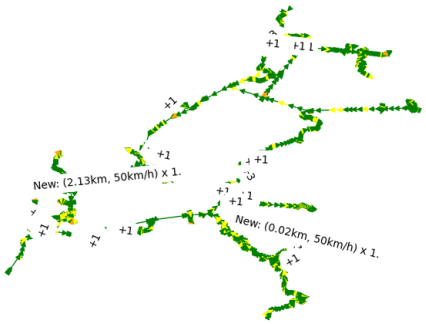
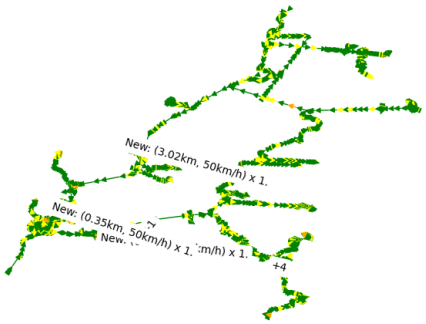
Colocação	Conj. parâmetros	Fitness	Layout
31º	Indivíduos - 10 Tx. Mutação - 1% Tx. Elitismo - 10% Tx. Filhos gerados - 50% Téc. Seleção - Aleatória	14.3549	
32º	Indivíduos - 10 Tx. Mutação - 10% Tx. Elitismo - 20% Tx. Filhos gerados - 50% Téc. Seleção - Aleatória	14.3525	
33º	Indivíduos - 10 Tx. Mutação - 1% Tx. Elitismo - 10% Tx. Filhos gerados - 50% Téc. Seleção - Torneio	14.3516	

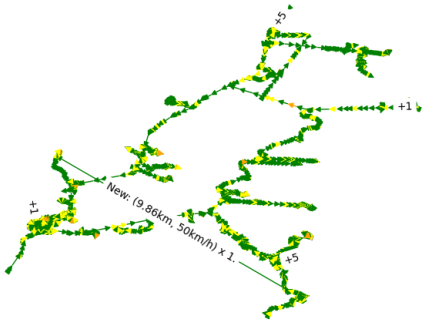
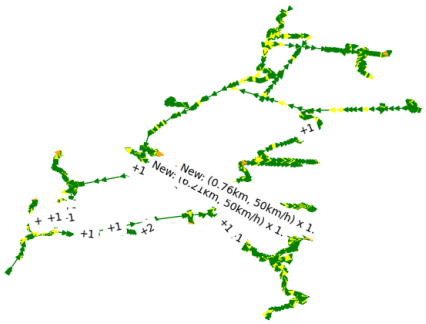
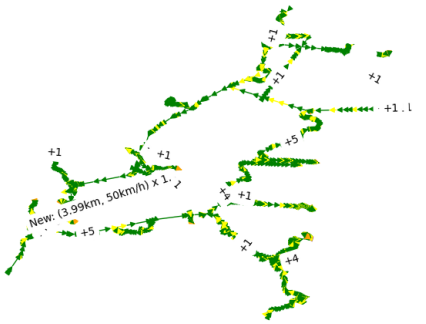
Colocação	**Conj. parâmetros**	**Fitness**	**Layout**
34º	Indivíduos - 10 Tx. Mutação - 10% Tx. Elitismo - 20% Tx. Filhos gerados - 25% Téc. Seleção - Torneio	14.3515	
35º	Indivíduos - 15 Tx. Mutação - 10% Tx. Elitismo - 10% Tx. Filhos gerados - 50% Téc. Seleção - Roleta	14.3509	
36º	Indivíduos - 20 Tx. Mutação - 10% Tx. Elitismo - 10% Tx. Filhos gerados - 25% Téc. Seleção - Aleatória	14.3503	

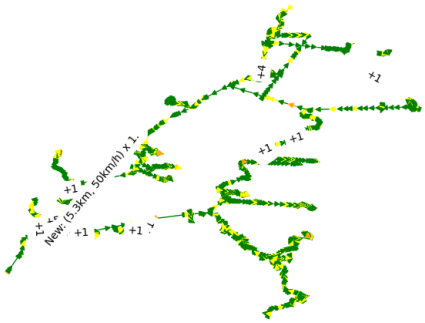
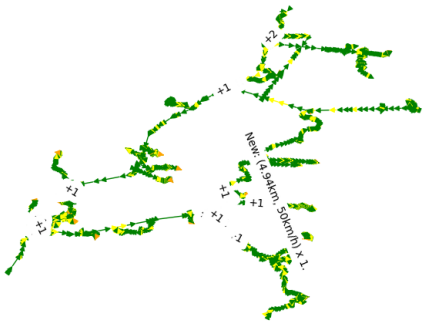
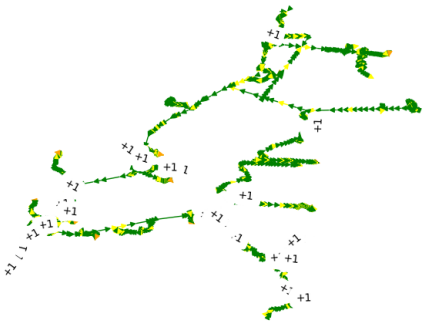
Colocação	Conj. parâmetros	Fitness	Layout
37º	Indivíduos - 10 Tx. Mutação - 1% Tx. Elitismo - 20% Tx. Filhos gerados - 50% Téc. Seleção - Roleta	14.3489	
38º	Indivíduos - 15 Tx. Mutação - 1% Tx. Elitismo - 10% Tx. Filhos gerados - 25% Téc. Seleção - Torneio	14.3430	
39º	Indivíduos - 20 Tx. Mutação - 1% Tx. Elitismo - 10% Tx. Filhos gerados - 50% Téc. Seleção - Roleta	14.3429	

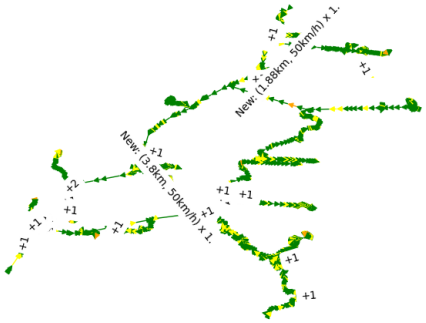
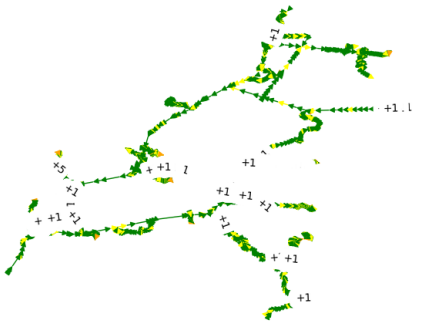
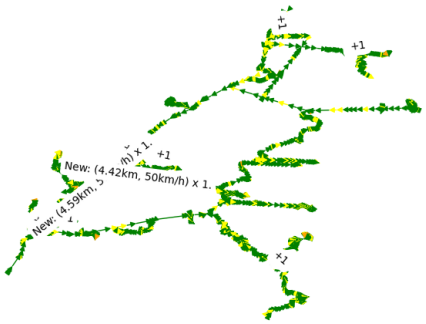
Colocação	**Conj. parâmetros**	**Fitness**	**Layout**
40º	Indivíduos - 15 Tx. Mutação - 10% Tx. Elitismo - 10% Tx. Filhos gerados - 50% Téc. Seleção - Aleatória	14.3406	
41º	Indivíduos - 10 Tx. Mutação - 1% Tx. Elitismo - 10% Tx. Filhos gerados - 50% Téc. Seleção - Roleta	14.3398	
42º	Indivíduos - 10 Tx. Mutação - 10% Tx. Elitismo - 20% Tx. Filhos gerados - 50% Téc. Seleção - Roleta	14.3395	

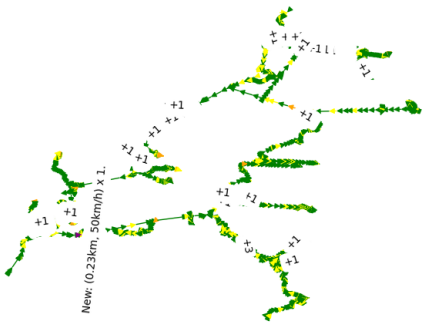
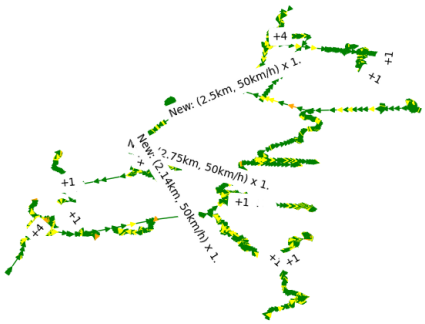
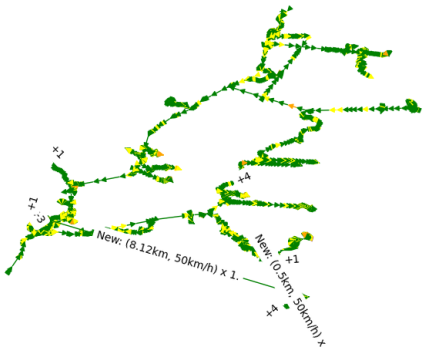
Colocação	**Conj. parâmetros**	**Fitness**	**Layout**
43º	Indivíduos - 10 Tx. Mutação - 1% Tx. Elitismo - 20% Tx. Filhos gerados - 25% Téc. Seleção - Roleta	14.3387	
44º	Indivíduos - 10 Tx. Mutação - 1% Tx. Elitismo - 20% Tx. Filhos gerados - 50% Téc. Seleção - Torneio	14.3382	
45º	Indivíduos - 20 Tx. Mutação - 10% Tx. Elitismo - 10% Tx. Filhos gerados - 50% Téc. Seleção - Aleatória	14.3372	

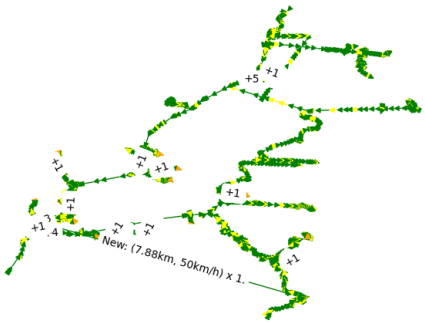
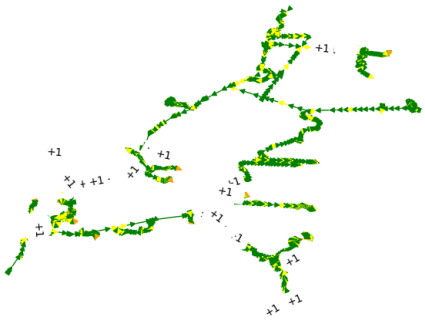
Colocação	**Conj. parâmetros**	**Fitness**	**Layout**
46º	Indivíduos - 10 Tx. Mutação - 1% Tx. Elitismo - 10% Tx. Filhos gerados - 25% Téc. Seleção - Torneio	14.3357	
47º	Indivíduos - 10 Tx. Mutação - 10% Tx. Elitismo - 10% Tx. Filhos gerados - 50% Téc. Seleção - Roleta	14.3356	
48º	Indivíduos - 20 Tx. Mutação - 10% Tx. Elitismo - 20% Tx. Filhos gerados - 50% Téc. Seleção - Aleatória	14.3335	

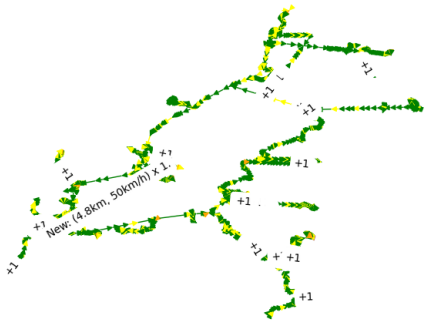
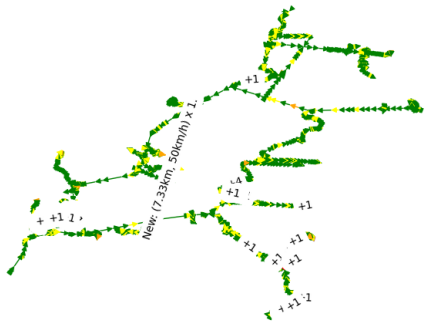
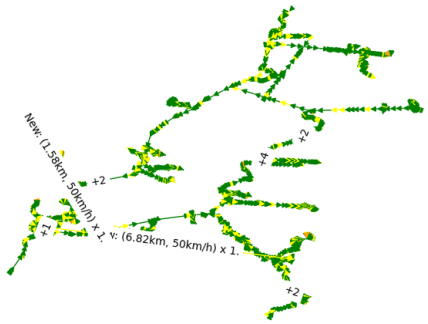
Colocação	**Conj. parâmetros**	**Fitness**	**Layout**
49º	Indivíduos - 15 Tx. Mutação - 10% Tx. Elitismo - 10% Tx. Filhos gerados - 25% Téc. Seleção - Roleta	14.3330	
50º	Indivíduos - 20 Tx. Mutação - 1% Tx. Elitismo - 20% Tx. Filhos gerados - 25% Téc. Seleção - Roleta	14.3326	
51º	Indivíduos - 15 Tx. Mutação - 1% Tx. Elitismo - 10% Tx. Filhos gerados - 25% Téc. Seleção - Aleatória	14.3311	

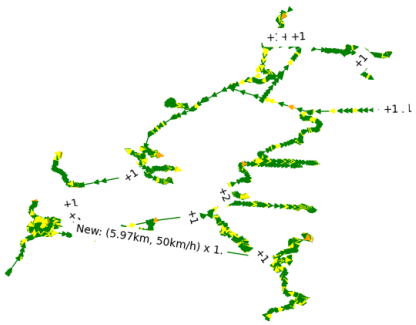
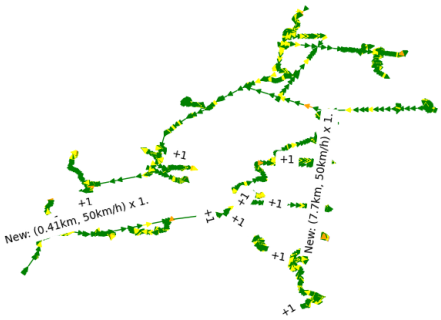
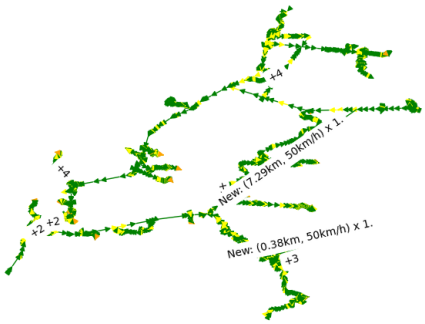
Colocação	Conj. parâmetros	Fitness	Layout
52º	Indivíduos - 15 Tx. Mutação - 10% Tx. Elitismo - 20% Tx. Filhos gerados - 50% Téc. Seleção - Roleta	14.3302	
53º	Indivíduos - 15 Tx. Mutação - 1% Tx. Elitismo - 20% Tx. Filhos gerados - 50% Téc. Seleção - Roleta	14.3299	
54º	Indivíduos - 20 Tx. Mutação - 1% Tx. Elitismo - 10% Tx. Filhos gerados - 25% Téc. Seleção - Aleatória	14.3299	

Colocação	Conj. parâmetros	Fitness	Layout
55º	Indivíduos - 20 Tx. Mutação - 10% Tx. Elitismo - 10% Tx. Filhos gerados - 50% Téc. Seleção - Roleta	14.3292	
56º	Indivíduos - 10 Tx. Mutação - 10% Tx. Elitismo - 10% Tx. Filhos gerados - 25% Téc. Seleção - Aleatória	14.3274	
57º	Indivíduos - 20 Tx. Mutação - 10% Tx. Elitismo - 20% Tx. Filhos gerados - 25% Téc. Seleção - Roleta	14.3269	

Colocação	**Conj. parâmetros**	**Fitness**	**Layout**
58º	Indivíduos - 10 Tx. Mutação - 1% Tx. Elitismo - 10% Tx. Filhos gerados - 25% Téc. Seleção - Aleatória	14.3234	
59º	Indivíduos - 15 Tx. Mutação - 1% Tx. Elitismo - 10% Tx. Filhos gerados - 50% Téc. Seleção - Aleatória	14.3215	
60º	Indivíduos - 10 Tx. Mutação - 10% Tx. Elitismo - 10% Tx. Filhos gerados - 50% Téc. Seleção - Aleatória	14.3212	

Colocação	Conj. parâmetros	Fitness	Layout
61º	Indivíduos - 10 Tx. Mutação - 1% Tx. Elitismo - 20% Tx. Filhos gerados - 25% Téc. Seleção - Aleatória	14.3210	
62º	Indivíduos - 15 Tx. Mutação - 1% Tx. Elitismo - 20% Tx. Filhos gerados - 25% Téc. Seleção - Aleatória	14.3197	
63º	Indivíduos - 20 Tx. Mutação - 1% Tx. Elitismo - 10% Tx. Filhos gerados - 25% Téc. Seleção - Roleta	14.3195	

Colocação	Conj. parâmetros	Fitness	Layout
64º	Indivíduos - 10 Tx. Mutação - 1% Tx. Elitismo - 20% Tx. Filhos gerados - 25% Téc. Seleção - Torneio	14.3179	
65º	Indivíduos - 10 Tx. Mutação - 10% Tx. Elitismo - 20% Tx. Filhos gerados - 25% Téc. Seleção - Aleatória	14.3143	
66º	Indivíduos - 10 Tx. Mutação - 10% Tx. Elitismo - 10% Tx. Filhos gerados - 25% Téc. Seleção - Roleta	14.3136	

Colocação	**Conj. parâmetros**	**Fitness**	**Layout**
67º	Indivíduos - 15 Tx. Mutação - 10% Tx. Elitismo - 20% Tx. Filhos gerados - 25% Téc. Seleção - Roleta	14.3134	
68º	Indivíduos - 15 Tx. Mutação - 10% Tx. Elitismo - 10% Tx. Filhos gerados - 25% Téc. Seleção - Aleatória	14.3125	
69º	Indivíduos - 10 Tx. Mutação - 1% Tx. Elitismo - 10% Tx. Filhos gerados - 25% Téc. Seleção - Roleta	14.3105	

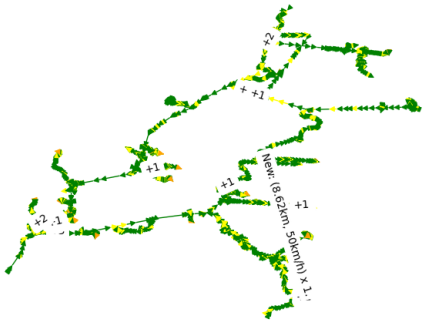
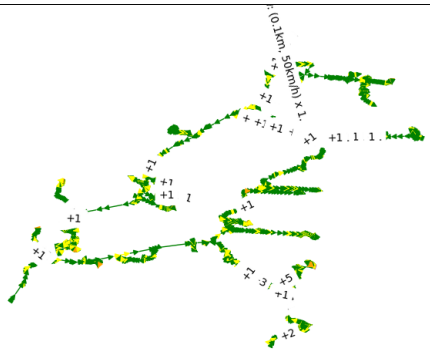
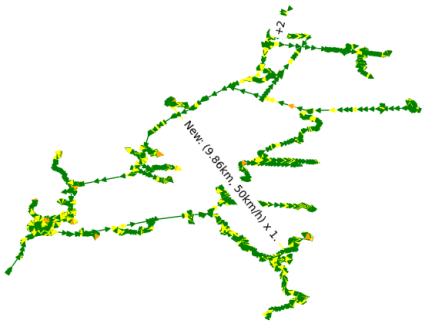
Colocação	**Conj. parâmetros**	**Fitness**	**Layout**
70º	Indivíduos - 20 Tx. Mutação - 10% Tx. Elitismo - 20% Tx. Filhos gerados - 25% Téc. Seleção - Aleatória	14.3097	
71º	Indivíduos - 10 Tx. Mutação - 1% Tx. Elitismo - 20% Tx. Filhos gerados - 50% Téc. Seleção - Aleatória	14.3056	
72º	Indivíduos - 10 Tx. Mutação - 10% Tx. Elitismo - 20% Tx. Filhos gerados - 25% Téc. Seleção - Roleta	14.3025	

Tabela 2 – Classificação com os melhores individuos de todos os conjuntos de parâmetros

6.2 Código-fonte

main.py

```
from load_place import create_net_file_from
from ga import run_genetic_algorithm
from problem_definition import ProblemDefinition

place_name = "Bora, Brazil"

file_name = create_net_file_from(place_name)
problem = ProblemDefinition(file_name)
run_genetic_algorithm(problem)
```

ga.py

```
import random
from individual import Individual
from problem_definition import ProblemDefinition

class GeneticAlgorithm:
    def __init__(self, problem):
        self.problem = problem
        self.generationSize = 10
        self.offspringSise = 5
        self.mutationRate = 0.001

    def mutationAddLane(self, ind, trackIds, chanceToStop=0.1):

        metersThreshold = self.problem.budgetUpdate * 1000.0 - ind.
    regularInsertedKMeters * 1000.0  # threshold of change to be designed
        metersReached = 0  # meters chaged

        # select a random node
        trackId = random.choice(trackIds)
        track = self.problem.tracks[trackId]
        trackIds.remove(trackId)
        trackList = []
        metersReached += track["distance"]
        node = track["t"]
        while metersReached < metersThreshold:
            trackList.append(trackId)
            if random.random() < chanceToStop:
                break
            # select a new destination
            if node in self.problem.adjLst:
```

```
30                 res = random.choice(list(self.problem.adjLst[node].items())
   )
31                 if res == None: break
32                 t = res[0]
33                 if t not in self.problem.adjLst[node]: break
34                 track = random.choice(self.problem.adjLst[node][t])
35                 if track == None: break
36                 trackId = track["id"]
37                 metersReached += track["distance"]
38                 node = track["t"]
39         # updating new lanes
40         for trackId in trackList:
41             ind.setLaneValue(str(trackId), +1)
42 
43     def mutationAddLaneOnRoad(self, ind, trackIds):
44 
45         metersThreshold = self.problem.budgetUpdate * 1000.0 - ind.
   regularInsertedKMeters * 1000.0  # threshold of change to be designed
46         metersReached = 0  # meters chaged
47 
48         trackList = []
49 
50         # repeat until find a road with more than one track
51         while len(trackList) <= 1:
52             trackList = []
53 
54             # select a random node
55             if len(trackIds) == 0: break
56             trackId = random.choice(trackIds)
57             track = self.problem.tracks[trackId]
58             metersReached = 0
59             trackIds.remove(trackId)
60 
61             # finding the road starting node
62             nodeStart = track["s"]
63             lastNode = track["t"]
64             if nodeStart not in self.problem.adjLst: break
65             while len(list(self.problem.adjLst[nodeStart])) == 1:
66                 if nodeStart != lastNode:
67                     track = self.problem.adjLst[nodeStart][lastNode][0]
68                     metersReached += track["distance"]
69                     trackId = int(track["id"])
70                     trackList.append(trackId)
71                     lastNode = nodeStart
72                     if nodeStart in self.problem.adjLstInv:
73                         nodeStart = next(iter(self.problem.adjLstInv[
   nodeStart]))
```

```
74                  else: break
75
76              # finding the road last node
77
78              lastNode = track["t"]
79              if lastNode not in self.problem.adjLst: break
80              while len(list(self.problem.adjLst[lastNode])) == 1:
81                  nextNode = next(iter(self.problem.adjLst[lastNode]))
82                  track = self.problem.adjLst[lastNode][nextNode][0]
83                  metersReached += track["distance"]
84                  trackId = int(track["id"])
85                  trackList.append(trackId)
86                  lastNode = nextNode
87                  if lastNode not in self.problem.adjLst: break
88
89          if metersReached < metersThreshold:
90              # updating new lanes
91              for trackId in trackList:
92                  ind.setLaneValue(str(trackId), +1)
93
94      def fitness(self, ind):
95          return None
96
97      def createInitialPopulation(self, size):
98          initialPopulation = list()
99          count = 0
100         while count < size:
101             initialPopulation.append(self.createNewRadomIndividual())
102             count += 1
103         return initialPopulation
104
105     def createNewRadomIndividual(self):
106         # create the individual
107         individual = Individual(self.problem)
108
109         # number of the iterations
110         count = random.randint(1, 100)
111
112         while count > 0:
113             # random selection of a track source
114             aleatoryTrackSource = str(random.randint(1, len(individual.
    tracks)))
115
116             # random selection of a track target
117             aleatoryTrackTarget = str(random.randint(1, len(individual.
    tracks)))
118
```

```
119             # random selection of a track id
120             aleatoryTrackId = str(random.randint(1, 1000))
121 
122             # random number for the option
123             randomNumber = random.randint(1, 3)
124             if randomNumber == 1:
125                 individual.insertAdditionalTrack(aleatoryTrackSource,
    aleatoryTrackTarget, 1500000, 1,
126                                                  self.problem.maxSpeed)
127             elif randomNumber == 2:
128                 individual.setLaneValue(aleatoryTrackId, random.randint(1,
    5))
129             else:
130                 individual.removeAdditionalTrack(aleatoryTrackSource,
    aleatoryTrackTarget, aleatoryTrackId)
131             count -= 1
132 
133         return individual
134 
135 def run_genetic_algorithm(problem):
136     best_individual_history = []
137     total_fitness_sum_history = []
138 
139     TOTAL_NUMBER_OF_INDIVIDUALS = 20
140     TOTAL_NUMBER_OF_GENERATIONS = 50
141     TOTAL_NUMBER_OF_ELITE_INDIVIDUALS = 2
142     TOTAL_NUMBER_OF_RANDOM_INDIVIDUALS = int(TOTAL_NUMBER_OF_INDIVIDUALS /
    2)
143     TOTAL_NUMBER_OF_MATED_INDIVIDUALS = TOTAL_NUMBER_OF_INDIVIDUALS -
    TOTAL_NUMBER_OF_ELITE_INDIVIDUALS - TOTAL_NUMBER_OF_RANDOM_INDIVIDUALS
144     SELECTION_TECHNIQUE = "torneio"
145     ga = GeneticAlgorithm(problem)
146     fitness_list = []
147     selected_parents = []
148     total_fitness_sum = 0
149     best_individual = None
150 
151     current_generation_individuals = ga.createInitialPopulation(
    TOTAL_NUMBER_OF_INDIVIDUALS)
152     new_generation_individuals = []
153 
154     #receives two Individuals, does crossover, mutation, and returns a new
    Individual
155     def mate(father, mother):
156         # do crossover
157         fatherModifications = father.getRandomModifications()
158         motherModifications = mother.getRandomModifications()
```

```
159
160         child = Individual(problem)
161         child.setModifications(fatherModifications, motherModifications)
162         mutation(child)
163
164         return child
165
166     def getAleatoryTrackIdsForMutation():
167         trackIds = []
168         while len(trackIds) < round((len(problem.tracks) - 1) * ga.
    mutationRate):
169             aleatoryId = random.randint(0, len(problem.tracks) - 1)
170             if aleatoryId not in trackIds:
171                 trackIds.append(aleatoryId)
172         return trackIds
173
174     def mutation(ind):
175         trackIds = getAleatoryTrackIdsForMutation()
176         while len(trackIds) > 0:
177             randomNumber = random.randint(1, 2)
178             if randomNumber == 1:
179                 ga.mutationAddLane(ind, trackIds)
180             elif randomNumber == 2:
181                 ga.mutationAddLaneOnRoad(ind, trackIds)
182         return ind
183
184
185     def draw_individual(selection_technique):
186         if selection_technique == "roleta":
187             random_number = random.randint(0, int(total_fitness_sum))
188
189             #stores the sum of the population fitness until the current
    individual
190             previous_fitness_sum = 0
191
192             # find winner individual
193             for i in range(TOTAL_NUMBER_OF_INDIVIDUALS):
194
195                 individual_fitness = fitness_list[i].fitness
196                 previous_fitness_sum += individual_fitness
197                 if previous_fitness_sum >= random_number:
198                     return current_generation_individuals[i]
199
200         elif selection_technique == "torneio":
201             # picks k random individuals from
    current_generation_individuals
202             competitors = random.choices(current_generation_individuals, k
```

```
    =5)
203
204             # returns competitor with max fitness
205             winner = max(competitors, key=lambda ind: ind.fitness)
206
207             return winner
208
209         elif selection_technique == "aleatorio":
210
211             # returns ANY individual, with no criteria
212             return random.choice(current_generation_individuals)
213
214     # runs the genetic algorithm on the number of generations specified
215     for gen in range(TOTAL_NUMBER_OF_GENERATIONS):
216
217         # fills the fitness_list and calculates total_fitness_sum
218         for i in range(TOTAL_NUMBER_OF_INDIVIDUALS):
219
220             individual = current_generation_individuals[i]
221             individual.calculateFitness()
222             fitness_list.append(individual)
223             total_fitness_sum += individual.fitness or 0
224
225         # we need to sort the fitness_list by their fitness in order to
    easily find the best individuals
226         sorted_fitness_list = sorted(fitness_list, key=lambda d: d.fitness,
     reverse=True)
227         best_individual = sorted_fitness_list[0]
228
229         total_fitness_sum_history.append(str(total_fitness_sum))
230         best_individual_history.append(str(best_individual.fitness) + "\t"
    + str(best_individual))
231
232
233         # selection / create selected_parents list
234         for i in range(TOTAL_NUMBER_OF_MATED_INDIVIDUALS):
235
236             father = draw_individual(SELECTION_TECHNIQUE)
237             mother = draw_individual(SELECTION_TECHNIQUE)
238
239             while mother == father:
240                 mother = draw_individual(SELECTION_TECHNIQUE)
241
242             selected_parents.append({
243                 "father": father,
244                 "mother": mother
245             })
```

```
246
247             # mate parents to create new generation
248             new_generation_individuals.append(mate(**selected_parents[i]))
249
250
251         # add elite individuals to generation_individuals
252         for i in range(TOTAL_NUMBER_OF_ELITE_INDIVIDUALS):
253             new_generation_individuals.append(sorted_fitness_list[i])
254
255
256         # add random_individuals to generation_individuals
257         new_generation_individuals.extend(ga.createInitialPopulation(
    TOTAL_NUMBER_OF_RANDOM_INDIVIDUALS))
258
259         #updates the current_population
260         current_generation_individuals = new_generation_individuals
261         new_generation_individuals = []
262         fitness_list = []
263         total_fitness_sum = 0
264         print("generation: " + str(gen))
265
266     print("\n\nTOTAL_FITNESS_SUM_HISTORY:\n\n"+"\n".join(
    total_fitness_sum_history))
267     print("\n\nBEST_INDIVIDUAL_HISTORY:\n\n"+"\n".join(
    best_individual_history))
268     best_individual.printDesign()
```

user_model.py

```
1 from utils import indentZero
2 from translator import roads, road
3
4 class UserModel:
5     def fitness(self):
6         return float("inf")
7
8 class UMSalmanAlaswad_I(UserModel):
9
10     def __init__(self, problem):
11         self.vehiclesByRoad = 7
12         self.problem = problem
13
14     def fitness(self, individual):
15         individual.vehiclesByRoad = {}
16
17         # Is it required to renormalize travel time? (tt_i)
```

```
18          minTT = self.problem.minTT
19          nminTT = minTT
20
21          for source in individual.newTracks:
22              for target in individual.newTracks[source]:
23                  for track in individual.newTracks[source][target]:
24                      speed = self.problem.maxSpeed  # kmh
25                      if "maxspeed" in track:
26                          speed = track["maxspeed"]
27                      if nminTT > track["distance"]:
28                          nminTT = track["distance"] / 1000.0 * (1 / speed)
29          if minTT > nminTT:
30              # update travel time information in self.problem instance /
    newroads
31              for i in range(len(self.problem.tracks)):
32                  if "maxspeed" in self.problem.tracks[i]:
33                      speed = track["maxspeed"]
34                  roads[i]["tt"] = (road["distance"] / 1000.0 * (1 / speed))
    / nminTT
35              for source in individual.newTracks:
36                  for target in individual.newTracks[source]:
37                      for i in range(len(individual.newTracks[source][target
    ])):
38                          speed = self.problem.maxSpeed  # kmh
39                          if "maxspeed" in individual.newTracks[source][
    target][i]:
40                              speed = track["maxspeed"]
41                          individual.newTracks[source][target][i]["tt"] = (
    individual.newTracks[source][target][i][
42
      "distance"] / 1000.0 * (
43
               1 / speed)) / nminTT
44
45          # calculating pmatriz
46          pMatrix = {}
47          for track in self.problem.tracks:  # pmatrix for regular tracks
48              rid = track["id"]
49              s = track["s"]
50              t = track["t"]
51              pMatrix[rid] = {}
52
53              is_t_source_adjList = True
54              is_t_source_adjIndividual = True
55
56              if t not in self.problem.adjLst:
57                  is_t_source_adjList = False
```

```
58              if t not in individual.newTracks:
59                  is_t_source_adjIndividual = False
60
61              lanes = 0
62              if is_t_source_adjList == True:
63                  for tl in self.problem.adjLst[t]:
64                      for track2 in self.problem.adjLst[t][tl]:
65                          speed = self.problem.maxSpeed  # kmh
66                          if "maxspeed" in track2:
67                              speed = track2["maxspeed"]
68                          lanes += (track2["lanes"] + individual.tracks[
    track2["id"]]) * (1 / speed)
69
70              if is_t_source_adjIndividual == True:
71                  for tl in individual.newTracks[t]:
72                      for track2 in individual.newTracks[t][tl]:
73                          speed = self.problem.maxSpeed  # kmh
74                          if "maxspeed" in track2:
75                              speed = track2["maxspeed"]
76                          lanes += track2["lanes"] * (1 / speed)
77
78              if is_t_source_adjList == True or is_t_source_adjIndividual ==
    True:
79                  speed = self.problem.maxSpeed  # kmh
80                  if "maxspeed" in track:
81                      speed = track["maxspeed"]
82                  if track["distance"] == 0.0:
83                      pMatrix[rid][rid] = 0.0
84                  else:
85                      pMatrix[rid][rid] = (track["tt"] - 1) / (track["tt"])
86
87              if is_t_source_adjList == True:
88                  for tl in self.problem.adjLst[t]:
89                      for track2 in self.problem.adjLst[t][tl]:
90                          rid2 = track2["id"]
91                          pMatrix[rid][rid2] = 0
92                          speed = self.problem.maxSpeed  # km/h
93                          if "maxspeed" in track2:
94                              speed = track2["maxspeed"]
95                          pMatrix[rid][rid2] += (1.0 - pMatrix[rid][rid]) * (
96                                          ((track2["lanes"] + individual.tracks[
    track2["id"]]) * (1 / speed)) / (lanes))
97
98              if is_t_source_adjIndividual == True:
99
100                 for tl in individual.newTracks[t]:
101                     for track2 in individual.newTracks[t][tl]:
```

```
102                         rid2 = track2["id"]
103                         pMatrix[rid][rid2] = 0
104                         speed = self.problem.maxSpeed  # km/h
105                         if "maxspeed" in track2:
106                             speed = track2["maxspeed"]
107                         pMatrix[rid][rid2] += (1.0 - pMatrix[rid][rid]) *
    ((track2["lanes"] * (1 / speed)) / (lanes))
108 
109             if is_t_source_adjList == False and is_t_source_adjIndividual
    == False:
110                 pMatrix[rid][rid] = 1
111 
112         for s in individual.newTracks:
113             for t in individual.newTracks[s]:  # pmatrix for non-regular
    tracks
114                 for track in individual.newTracks[s][t]:  # pmatrix for non
    -regular tracks
115                     rid = track["id"]
116 
117                     pMatrix[rid] = {}
118 
119                     is_t_source_adjList = True
120                     is_t_source_adjIndividual = True
121 
122                     if t not in self.problem.adjLst:
123                         is_t_source_adjList = False
124                     if t not in individual.newTracks:
125                         is_t_source_adjIndividual = False
126 
127                     lanes = 0
128                     if is_t_source_adjList == True:
129                         for tl in self.problem.adjLst[t]:
130                             for track2 in self.problem.adjLst[t][tl]:
131                                 speed = self.problem.maxSpeed  # kmh
132                                 if "maxspeed" in track2:
133                                     speed = track2["maxspeed"]
134                                 lanes += (track2["lanes"] + individual.
    tracks[track2["id"]]) * (1 / speed)
135 
136                     if is_t_source_adjIndividual == True:
137                         for tl in individual.newTracks[t]:
138                             for track2 in individual.newTracks[t][tl]:
139                                 speed = self.problem.maxSpeed  # kmh
140                                 if "maxspeed" in track2:
141                                     speed = track2["maxspeed"]
142                                 lanes += track2["lanes"] * (1 / speed)
143 
```

```
144                     if is_t_source_adjList == True or
    is_t_source_adjIndividual == True:
145                         speed = self.problem.maxSpeed  # kmh
146                         if "maxspeed" in track:
147                             speed = track["maxspeed"]
148                         if track["distance"] == 0.0:
149                             pMatrix[rid][rid] = 0.0
150                         else:
151                             pMatrix[rid][rid] = (track["tt"] - 1) / (track
    ["tt"])
152 
153                     if is_t_source_adjList == True:
154                         for tl in self.problem.adjLst[t]:
155                             for track2 in self.problem.adjLst[t][tl]:
156                                 rid2 = track2["id"]
157                                 pMatrix[rid][rid2] = 0
158                                 speed = self.problem.maxSpeed  # km/h
159                                 if "maxspeed" in track2:
160                                     speed = track2["maxspeed"]
161                                 pMatrix[rid][rid2] += (1.0 - pMatrix[rid][
    rid]) * (
162                                                       ((track2["lanes"] + individual.
    tracks[track2["id"]]) * (1 / speed)) / (
163                                                           lanes))
164 
165                     if is_t_source_adjIndividual == True:
166                         for tl in individual.newTracks[t]:
167                             for track2 in individual.newTracks[t][tl]:
168                                 rid2 = track2["id"]
169                                 pMatrix[rid][rid2] = 0
170                                 speed = self.problem.maxSpeed  # km/h
171                                 if "maxspeed" in track2:
172                                     speed = track2["maxspeed"]
173                                 pMatrix[rid][rid2] += (1.0 - pMatrix[rid][
    rid]) * (
174                                                       (track2["lanes"] * (1 / speed))
     / (lanes))
175 
176                     if is_t_source_adjList == False and
    is_t_source_adjIndividual == False:
177                         pMatrix[rid][rid] = 1
178 
179         # number of vehicles by track
180         individual.vehiclesByRoad = {}
181         for i in range(len(self.problem.tracks)):  # number of vehicles for
     regular tracks
182             track = self.problem.tracks[i]
```

```
183             rid = track["id"]
184             s = track["s"]
185             t = track["t"]
186             if rid not in individual.vehiclesByRoad:
187                 individual.vehiclesByRoad[rid] = (self.vehiclesByRoad * (
    track["distance"] / 1000.0) * (
188                         track["lanes"] + individual.tracks[rid])) *
    pMatrix[rid][rid]
189 
190             is_t_source_adjList = True
191             is_t_source_adjIndividual = True
192 
193             if t not in self.problem.adjLst:
194                 is_t_source_adjList = False
195             if t not in individual.newTracks:
196                 is_t_source_adjIndividual = False
197 
198             if is_t_source_adjList == True:
199                 for tl in self.problem.adjLst[t]:
200                     for track2 in self.problem.adjLst[t][tl]:
201                         rid2 = track2["id"]
202                         if rid2 not in individual.vehiclesByRoad:
203                             individual.vehiclesByRoad[rid2] = (self.
    vehiclesByRoad * (track2["distance"] / 1000.0) * (
204                                     track2["lanes"] + individual.tracks
    [rid2])) * pMatrix[rid2][rid2]
205                         individual.vehiclesByRoad[rid2] += pMatrix[rid][
    rid2] * (
206                                     self.vehiclesByRoad * (track["distance
    "] / 1000.0) * (
207                                     track["lanes"] + individual.tracks[
    rid]))
208             if is_t_source_adjIndividual == True:
209                 for tl in individual.newTracks[t]:
210                     for track2 in individual.newTracks[t][tl]:
211                         rid2 = track2["id"]
212                         if rid2 not in individual.vehiclesByRoad:
213                             individual.vehiclesByRoad[rid2] = (self.
    vehiclesByRoad * (track2["distance"] / 1000.0) * (
214                             track2["lanes"])) * pMatrix[rid2][rid2]
215                         individual.vehiclesByRoad[rid2] += pMatrix[rid][
    rid2] * (
216                                     self.vehiclesByRoad * (track["distance
    "] / 1000.0) * (
217                                     track["lanes"] + individual.tracks[
    rid]))
218 
```

```
219         # for i in range(len(individual.newTracks)):
220         for s in individual.newTracks:  # number of vehicles for non-
    regular tracks
221             for t in individual.newTracks[s]:
222                 for track in individual.newTracks[s][t]:
223                     rid = track["id"]
224 
225                     if rid not in individual.vehiclesByRoad:
226                         individual.vehiclesByRoad[rid] = (self.
    vehiclesByRoad * (track["distance"] / 1000.0) * (
227                         track["lanes"])) * pMatrix[rid][rid]
228 
229                     is_t_source_adjList = True
230                     is_t_source_adjIndividual = True
231 
232                     if t not in self.problem.adjLst:
233                         is_t_source_adjList = False
234                     if t not in individual.newTracks:
235                         is_t_source_adjIndividual = False
236 
237                     if is_t_source_adjList == True:
238                         for tl in self.problem.adjLst[t]:
239                             for track2 in self.problem.adjLst[t][tl]:
240                                 rid2 = track2["id"]
241                                 if rid2 not in individual.vehiclesByRoad:
242                                     individual.vehiclesByRoad[rid2] = (self
    .vehiclesByRoad * (
243                                                 track2["distance"] /
    1000.0) * (track2["lanes"] + individual.tracks[
244                                             rid2])) * pMatrix[rid2][rid2]
245                                 individual.vehiclesByRoad[rid2] += pMatrix[
    rid][rid2] * (
246                                             self.vehiclesByRoad * (track["
    distance"] / 1000.0) * (track["lanes"]))
247                     if is_t_source_adjIndividual == True:
248                         for tl in individual.newTracks[t]:
249                             for track2 in individual.newTracks[t][tl]:
250                                 rid2 = track2["id"]
251                                 if rid2 not in individual.vehiclesByRoad:
252                                     individual.vehiclesByRoad[rid2] = (self
    .vehiclesByRoad * (
253                                                 track2["distance"] /
    1000.0) * (track2["lanes"])) * pMatrix[rid2][rid2]
254                                 individual.vehiclesByRoad[rid2] += pMatrix[
    rid][rid2] * (
255                                             self.vehiclesByRoad * (track["
    distance"] / 1000.0) * (track["lanes"]))
```

```
256
257         # fitnes value: average density
258         densSum = 0.0
259         densCount = 0
260         for track in self.problem.tracks:  # regular tracks
261             rid = track["id"]
262             dens = indentZero(individual.vehiclesByRoad[rid],
263                               ((track["distance"] / 1000.0) * (track["
    lanes"] + individual.tracks[rid])))
264             densSum += dens
265             densCount += 1
266         for s in individual.newTracks:
267             for t in individual.newTracks[s]:
268                 for track in individual.newTracks[s][t]:
269                     rid = track["id"]
270                     dens = indentZero(individual.vehiclesByRoad[rid],
271                                       ((track["distance"] / 1000.0) *
    (track["lanes"])))
272                     densSum += dens
273                     densCount += 1
274
275         individual.fitness = densCount * 100 / densSum
276         return individual.fitness
```

problem_definition.py

```
1 import copy
2 from math import radians, cos, sin, asin, sqrt
3
4 # reads the .net file from load_place.py and turns it into a graph
5 class ProblemDefinition:
6
7     def __init__(self, inputFile):
8
9         self.maxSpeed = 50  # in km/h
10        self.nodes = None
11        self.tracks = None
12        self.adjLst = None
13        self.adjLstInv = None
14        self.minTT = None
15        self.budgetUpdate = 10  # in km
16        self.budgetAddition = 10  # in km
17
18        self.nodes = {}
19        self.tracks = []
20
```

```
21          osmNetFile = open(inputFile, 'r')
22          lines = osmNetFile.readlines()
23
24          arcsPhase = False
25          roadId = 0
26          for line in lines:
27              if line[0:4] == '*arc':
28                  arcsPhase = True
29                  continue
30              data = line.split(' ')
31              data[-1] = data[-1].replace('\n', '')
32              if not arcsPhase:
33                  # node phase
34                  # 1 90199676 -48.54448 -27.6634265 ellipse
35                  if len(data) < 4: continue
36                  nodeId = data[0]
37                  nodeLong = float(data[2])
38                  nodeLat = float(data[3])
39                  word = ""
40                  for i in range(5, len(data)):
41                      sub = data[i]
42                      word += str(sub) + " "
43                  self.nodes[nodeId] = {"id": int(nodeId), "lat": nodeLat, "
    long": nodeLong, "properties": word.strip()}
44              else:
45                  # arc phase
46                  # 9 9499 1.0 lanes 3 highway trunk name "Rodovia Governador
     Aderbal Ramos da Silva" maxspeed 80 ref SC-401 oneway yes
47                  s = data[0]
48                  t = data[1]
49                  distance = max(0.1, self.haversine(self.nodes[s]["long"],
    self.nodes[s]["lat"], self.nodes[t]["long"],
50                                                     self.nodes[t]["lat"]))
51                  # arc metadata
52                  arc = {"s": s, "t": t, "distance": distance}
53                  for i in range(2, len(data)):
54                      word = data[i]
55                      if word == "oneway":
56                          arc["oneway"] = data[i + 1]
57                      if word == "lanes":
58                          arc["lanes"] = int(data[i + 1])
59                      if word == "maxspeed":
60                          arc["maxspeed"] = int(data[i + 1])
61                      if word == "ref":
62                          if data[i + 1][0] == "\"":
63                              name = ""
64                              c = 0
```

```
65                              for subw in data[i + 1:]:
66                                  c += 1
67                                  name += subw.replace("\"", "") + " "
68                                  if subw[-1] == "\"":
69                                      break
70                              arc["ref"] = name.strip()
71                              i += c - 1
72                              continue
73                          else:
74                              arc["ref"] = data[i + 1]
75                      if word == "name":
76                          name = ""
77                          c = 0
78                          for subw in data[i + 1:]:
79                              c += 1
80                              name += subw.replace("\"", "") + " "
81                              if subw[-1] == "\"":
82                                  break
83                          arc["name"] = name.strip()
84                          i += c - 1
85                          continue
86                  if "oneway" not in arc:
87                      arc["oneway"] = "yes"
88                  if "lanes" not in arc:
89                      arc["lanes"] = 1
90
91                  if arc["lanes"] > 1 and arc["oneway"] == "no":
92                      arc["lanes"] = max(1, int(arc["lanes"] // 2))
93                      arc2 = copy.deepcopy(arc)
94                      arc2["s"] = arc["t"]
95                      arc2["t"] = arc["s"]
96
97                      arc2["id"] = str(roadId)
98                      roadId += 1
99                      self.tracks.append(arc2)
100
101                 arc["id"] = str(roadId)
102                 roadId += 1
103                 self.tracks.append(arc)
104
105         # adjacent list
106         self.adjLst = {}
107         for road in self.tracks:
108             s = road["s"]
109             t = road["t"]
110             if s not in self.adjLst:
111                 self.adjLst[s] = {}
```

```
112             if t not in self.adjLst[s]:
113                 self.adjLst[s][t] = []
114             self.adjLst[s][t].append(road)
115 
116         # adjacent list inverted
117         self.adjLstInv = {}
118         for road in self.tracks:
119             s = road["s"]
120             t = road["t"]
121             if t not in self.adjLstInv:
122                 self.adjLstInv[t] = {}
123             if s not in self.adjLstInv[t]:
124                 self.adjLstInv[t][s] = []
125             self.adjLstInv[t][s].append(road)
126 
127         # normalizing travel time (tt_i)
128         self.minTT = float("+inf")
129         for road in self.tracks:
130             speed = self.maxSpeed  # kmh
131             if "maxspeed" in road:
132                 speed = road["maxspeed"]
133             if self.minTT > road["distance"]:
134                 self.minTT = road["distance"] / 1000.0 * (1 / speed)
135         for i in range(len(self.tracks)):
136             road = self.tracks[i]
137             speed = self.maxSpeed  # kmh
138             if "maxspeed" in road:
139                 speed = road["maxspeed"]
140             self.tracks[i]["tt"] = (self.tracks[i]["distance"] / 1000.0 *
    (1 / speed)) / self.minTT
141 
142     def haversine(self, lon1, lat1, lon2, lat2):
143         """
144         Calculate the great circle distance between two points
145         on the earth (specified in decimal degrees)
146         """
147         # convert decimal degrees to radians
148         lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])
149         # haversine formula
150         dlon = lon2 - lon1
151         dlat = lat2 - lat1
152         a = sin(dlat / 2) ** 2 + cos(lat1) * cos(lat2) * sin(dlon / 2) ** 2
153         c = 2 * asin(sqrt(a))
154         # Radius of earth in kilometers is 6371
155         km = 6371 * c
156         return km * 1000.0  # in meters
```

translator.py

```
import osmnx as ox
import networkx as nx
import copy
import matplotlib.pyplot as plt
import utm

from math import radians, cos, sin, asin, sqrt

def haversine(lon1, lat1, lon2, lat2):
    """
    Calculate the great circle distance between two points
    on the spherical surface of Earth (specified in decimal degrees)
    """
    # convert decimal degrees to radians
    lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])
    # haversine formula
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
    c = 2 * asin(sqrt(a))
    # Radius of earth in kilometers is 6371
    km = 6371* c
    return km * 1000.0

place_name =  "Bora, Brazil"
inputFile = place_name+'.net'

nodes = {}
roads = []

osmNetFile = open(inputFile, 'r')
lines = osmNetFile.readlines()

arcsPhase = False
roadId= 0
for line in lines:
    if line[0:4] == '*arc':
        arcsPhase = True
        continue
    data = line.split(' ')
    data[-1] = data[-1].replace('\n','')
    if not arcsPhase:
        #node phase
        #1 90199676 -48.54448 -27.6634265 ellipse
        if len(data) < 4: continue
```

```
46          nodeId = data[0]
47          nodeLong = float(data[2])
48          nodeLat = float(data[3])
49          word = ""
50          for i in range(5, len(data)):
51              sub = data[i]
52              word += str(sub) + " "
53          nodes[nodeId] = {"id": int(nodeId), "lat": nodeLat, "long":nodeLong
    , "properties":word.strip()}
54      else:
55          #arc phase
56          #9 9499 1.0 lanes 3 highway trunk name "Rodovia Governador Aderbal
    Ramos da Silva" maxspeed 80 ref SC-401 oneway yes
57          s = data[0]
58          t = data[1]
59          distance = max(0.1,haversine(nodes[s]["long"],nodes[s]["lat"],nodes
    [t]["long"],nodes[t]["lat"]))
60          #arc metadata
61          arc={"s": s, "t": t, "distance": distance}
62          for i in range(2, len(data)):
63              word = data[i]
64              if word == "oneway":
65                  arc["oneway"] = data[i+1]
66              if word == "lanes":
67                  arc["lanes"] = int (data[i+1])
68              if word == "maxspeed":
69                  arc["maxspeed"] = int (data[i+1])
70              if word == "ref":
71                  if data[i+1][0] == "\"":
72                      name = ""
73                      c=0
74                      for subw in data[i+1:]:
75                          c+=1
76                          name += subw.replace("\"","")+" "
77                          if subw[-1] == "\"":
78                              break
79                      arc["ref"] = name.strip()
80                      i+= c -1
81                      continue
82                  else:
83                      arc["ref"] = data[i+1]
84              if word == "name":
85                  name = ""
86                  c=0
87                  for subw in data[i+1:]:
88                      c+=1
89                      name += subw.replace("\"","")+" "
```

```
                    if subw[-1] == "\"":
                        break
                arc["name"] = name.strip()
                i+= c -1
                continue
        if "oneway" not in arc:
            arc["oneway"] = "yes"
        if "lanes" not in arc:
            arc["lanes"] = 1

        if arc["lanes"] > 1 and arc["oneway"] == "no":
            arc["lanes"] = max(1, int(arc["lanes"] // 2))
            arc2 = copy.deepcopy(arc)
            arc2["s"] = arc["t"]
            arc2["t"] = arc["s"]
            roadId+=1
            arc2["id"] = str(roadId)
            roads.append(arc2)
        roadId+=1
        arc["id"] = str(roadId)
        roads.append(arc)

#adjacent list
adjLst = {}
for road in roads:
    s = road["s"]
    t = road["t"]
    if s not in adjLst:
        adjLst[s]={}
    if t not in adjLst[s]:
        adjLst[s][t] = []
    adjLst[s][t].append(road)

maxSpeed = 50

#normalizing travel time (tt_i)
minTT = float("+inf")
for road in roads:
    speed = maxSpeed #kmh
    if "maxspeed" in road:
        speed = road["maxspeed"]
    if minTT > road["distance"]:
        minTT = road["distance"]/1000.0 * (1/speed)
for i in range(len(roads)):
    roads[i]["tt"] = (road["distance"]/1000.0 * (1/speed)) / minTT

minimum = float("+inf")
```

```
137
138 for road in roads:
139     speed = maxSpeed #kmh
140     if "maxspeed" in road:
141         if maxSpeed< road["maxspeed"]: maxSpeed = road["maxspeed"]
142         speed = road["maxspeed"]
143     value = road["distance"]/1000.0 * (1/speed)
144     if minimum > value:
145         minimum = value
146
147 pMatrix = {}
148 for road in roads:
149     rid = road["id"]
150     s = road["s"]
151     t = road["t"]
152     pMatrix[rid] = {}
153
154     if t not in adjLst:
155         pMatrix[rid][rid] = 1
156         continue
157
158     lanes = 0
159     for tl in adjLst[t]:
160         for road2 in adjLst[t][tl]:
161             speed = maxSpeed #kmh
162             if "maxspeed" in road2:
163                 speed = road2["maxspeed"]
164             lanes += road2["lanes"]*(1/speed)
165
166     speed = maxSpeed #kmh
167     if "maxspeed" in road:
168         speed = road["maxspeed"]
169     if road["distance"] == 0.0:
170         pMatrix[rid][rid] = 0.0
171     else:
172
173         pMatrix[rid][rid] = (road["tt"]-1) / (road["tt"])
174         #pMatrix[rid][rid] = ((1/speed) - 1/maxSpeed)/ ((1/speed))
175         #pMatrix[rid][rid] = (road["distance"]/1000.0 * (1/speed) - 1/60)/
    (road["distance"]/1000.0 * (1/speed))
176         #pMatrix[rid][rid] = ((road["lanes"] * (150-speed) - value)/ lanes)
177
178     for tl in adjLst[t]:
179         for road2 in adjLst[t][tl]:
180             rid2 = road2["id"]
181             pMatrix[rid][rid2] = 0
182             speed = maxSpeed #kmh
```

```
183             if "maxspeed" in road2:
184                 speed = road2["maxspeed"]
185             pMatrix[rid][rid2] += (1.0-pMatrix[rid][rid])*((road2["lanes"]
    * (1/speed))/ (lanes) )
186 
187 #distributing vehicles
188 numVehicles = 7 #por km por pista
189 vehiclesByRoad = {}
190 for i in range(len(roads)):
191     road = roads[i]
192     rid = road["id"]
193     s = road["s"]
194     t = road["t"]
195     if rid not in vehiclesByRoad:
196         vehiclesByRoad[rid] = (numVehicles*(road["distance"]/1000.0)*road["
    lanes"])*pMatrix[rid][rid]
197     if t not in adjLst:
198         continue
199     for tl in adjLst[t]:
200         for road2 in adjLst[t][tl]:
201             rid2 = road2["id"]
202             if rid2 not in vehiclesByRoad:
203                 vehiclesByRoad[rid2] = (numVehicles*(road2["distance
    "]/1000.0)*road2["lanes"])*pMatrix[rid2][rid2]
204             vehiclesByRoad[rid2] += pMatrix[rid][rid2]*(numVehicles*(road["
    distance"]/1000.0)*road["lanes"])
205 
206 maxDensity = 0
207 maxId = []
208 for i in range(len(roads)):
209     road = roads[i]
210     rid = road["id"]
211     roads[i]["dens"] = vehiclesByRoad[rid]
212     if vehiclesByRoad[rid] > maxDensity:
213         maxDensity = vehiclesByRoad[rid]
214         maxId.clear()
215     if vehiclesByRoad[rid] == maxDensity:
216         maxId.append(road)
217 
218 print(maxId)
219 print(maxDensity)
220 
221 edgeColors = []
222 for road in roads:
223     color = ""
224     if road["dens"] <= 7*(road["distance"]/1000.0)*road["lanes"]:
225         color="green"
```

```
226     elif road["dens"] <= 11*(road["distance"]/1000.0)*road["lanes"]:
227         color="yellow"
228     elif road["dens"] <= 22*(road["distance"]/1000.0)*road["lanes"]:
229         color="orange"
230     elif road["dens"] <= 28*(road["distance"]/1000.0)*road["lanes"]:
231         color="purple"
232     else:
233         color="red"
234     edgeColors.append(color)
235 
236 #creating graph for presentation
237 graph = nx.MultiGraph()
238 
239 i=0
240 for node in nodes:
241     graph.add_node(str(nodes[node]["id"]))
242 
243 #arcs
244 for road in roads:
245     #if road["s"] not in graph.nodes() or road["t"] not in graph.nodes():
        continue
246     graph.add_edge(road["s"], road["t"])
247 
248 nodePos={}
249 #for node in nodes:
250 for (node, data) in graph.nodes(data=True):
251     l = utm.from_latlon(nodes[node]["lat"], nodes[node]["long"])
252     x, y =  l[0], l[1]
253     nodePos[node] = [x,y]
254 
255 nx.draw_networkx(graph, pos=nodePos,  with_labels=False, node_size=0,
        edge_color=edgeColors)
256 plt.savefig(place_name+".png", format="PNG")
257 plt.savefig(place_name+".pdf", format="PDF")
258 #plt.show()
259 
260 
261 #nx.draw_networkx(graph, pos=nodePos, with_labels=True, edge_color=
        edgeColors)
262 #plt.tight_layout()
263 #plt.savefig("output.pdf", format="PDF")
264 
265 #plt.savefig("output.png", format="PNG")
266 #nx.draw(graph, edge_color=edgeColors)
267 #fig, ax = ox.plot_graph(graph, edge_color=edgeColors)
```

load_place.py

```
import osmnx as ox
import networkx as nx

import matplotlib.pyplot as pyplot

useful_tags_node = ox.settings.useful_tags_node
osm_xml_node_attrs = ox.settings.osm_xml_node_attrs
osm_xml_node_tags = ox.settings.osm_xml_node_tags
useful_tags_way = ox.settings.useful_tags_way
osm_xml_way_attrs = ox.settings.osm_xml_way_attrs
osm_xml_way_tags = ox.settings.osm_xml_way_tags

config_parameters = {
    "useful_tags_node": list(set(useful_tags_node + osm_xml_node_attrs +
    osm_xml_node_tags)),
    "useful_tags_way": list(set(useful_tags_way + osm_xml_way_attrs +
    osm_xml_way_tags)),
    "all_oneway": True
}

def create_net_file_from(place_name):
    ox.config(**config_parameters)

    outputName=place_name #.replace(" ", "_")

    # loads driveable streets as a graph from the place name
    # more in: https://geoffboeing.com/2016/11/osmnx-python-street-networks
    /
    graph = ox.graph_from_place(place_name, simplify=False,  network_type='
    drive')

    file_name = outputName+".net"

    # converts "graph" to a simpler .net file
    nx.write_pajek(graph, file_name)

    # nx.write_graphml(graph, "output.graphml")
    # ox.save_graph_xml(graph, filepath='output.osm')
    # fig, ax = ox.plot_graph(graph)

    #pyplot.tight_layout()
    return file_name
```

individual.py

```
import osmnx as ox
from user_model import UMSalmanAlaswad_I
import random
import networkx as nx
import matplotlib.pyplot as pyplot
import utm

#reads the problem (a city graph) and creates new attributes/functions
    useful for a genetic algorithm
class Individual:

    def __init__(self, problem):

        self.userModel = UMSalmanAlaswad_I(problem)
        self.problem = problem
        self.fitness = None
        self.vehiclesByRoad = None
        # variable for crossover
        # for the tracks with new number of lanes
        self.newLanes = {}
        self.tracks = {}  # positive or negative number of adapted lanes
        self.newTracks = {}
        # for preexisting tracks
        self.regularInsertedKMeters = 0
        self.regularRemovedKMeters = 0
        # for new tracks
        self.newInsertedKMeters = 0
        # id for new tracks
        self.nextTracksId = 0
        # no influences on the number of lanes for each track
        for track in problem.tracks:
            self.tracks[track["id"]] = 0


    #return the vehicles by roads
    def getVehiclesByRoad(self):
        return self.vehiclesByRoad

    #return the Inserted Km in new tracks
    def getNewInsertedKMeters(self):
        return self.newInsertedKMeters

    #return the number of the new tracks
    def getNewTracksSize(self):
        return len(self.newTracks)

    #return the number of adapted tracks
```

```
47      def getTracksSize(self):
48          return len(self.tracks)
49
50      #return the next track id
51      def getNextTracksId(self):
52          return self.nextTracksId
53
54  # return the number of adaptation on a track
55      def getLaneValue(self, tid):
56          return self.tracks[tid]
57
58      # method for crossover
59      def getModifications(self):
60          return {
61              "newTracks": self.newTracks,
62              "newLanes": self.newLanes
63          }
64
65      # method for crossover
66      # each modification has a 50% of being returned
67      def getRandomModifications(self):
68          modifications = self.getModifications()
69
70          randomNewTracks = self.getAleatoryNewTracksModifications(
    modifications["newTracks"])
71          randomNewLanes = self.getAleatoryNewLanesModifications(
    modifications["newLanes"])
72
73          return {
74              "newTracks": randomNewTracks,
75              "newLanes": randomNewLanes
76          }
77
78      def getAleatoryNewTracksModifications(self, modifications):
79          randomNewTracks = modifications.copy()
80          for i in range(len(modifications)):
81              randomKeyForNewTracks = random.choice(list(randomNewTracks.keys
    ()))
82              if random.random() < 0.5:
83                  randomNewTracks.pop(randomKeyForNewTracks)
84          return randomNewTracks
85
86      def getAleatoryNewLanesModifications(self, modifications):
87          randomNewLanes = modifications.copy()
88          for i in range(len(modifications)):
89              randomKeyForNewLanes = random.choice(list(randomNewLanes.keys()
    ))
```

```
90              if random.random() < 0.5:
91                  randomNewLanes.pop(randomKeyForNewLanes)
92          return randomNewLanes
93
94      # method for crossover
95      def setModifications(self, fatherModifications, motherModifications):
96          newTracks = {**fatherModifications["newTracks"], **
    motherModifications["newTracks"]}
97          newLanes = {**fatherModifications["newLanes"], **
    motherModifications["newLanes"]}
98
99          for source in newTracks.keys():
100             for target in newTracks[source].keys():
101                 for track in newTracks[source][target]:
102                     self.insertAdditionalTrack(track["s"], track["t"],
    track["distance"], track["lanes"], track["maxspeed"])
103
104         for tid, newNoLanes in newLanes.items():
105             self.setLaneValue(tid, newNoLanes)
106
107         self.resetFitness()
108
109     # set the number of lanes adapted in a track
110     def setLaneValue(self, tid, newNoLanes):
111         i = int(tid)
112         if self.tracks[tid] > 0:
113             self.regularInsertedKMeters -= self.tracks[tid] * self.problem.
    tracks[i]["distance"] / 1000.0
114         if self.tracks[tid] < 0:
115             self.regularRemovedKMeters -= self.tracks[tid] * self.problem.
    tracks[i]["distance"] / 1000.0
116         self.tracks[tid] = newNoLanes
117         self.newLanes[tid] = newNoLanes
118         if newNoLanes > 0:
119             self.regularInsertedKMeters += newNoLanes * self.problem.tracks
    [i]["distance"] / 1000.0
120         if newNoLanes < 0:
121             self.regularRemovedKMeters += newNoLanes * self.problem.tracks[
    i]["distance"] / 1000.0
122
123         self.resetFitness()
124
125     #return the fitness metric
126     def calculateFitness(self):
127         self.fitness = self.userModel.fitness(self)
128         return self.fitness
129
```

```
130     def resetFitness(self):
131         self.fitness = None
132 
133     # insert a new nonregular track between two nodes
134     def insertAdditionalTrack(self, source, target, distance, lanes,
    maxspeed):
135         if source not in self.newTracks:
136             self.newTracks[source] = {}
137         if target not in self.newTracks[source]:
138             self.newTracks[source][target] = []
139         self.nextTracksId += 1
140         tt = (distance / 1000.0 * (1 / maxspeed)) / self.problem.minTT
141         self.newTracks[source][target].append(
142             {"id": "nt_" + str(self.nextTracksId), "s": source, "t": target
    , "distance": distance, "lanes": lanes,
143              "maxspeed": maxspeed, "tt": tt})
144         self.newInsertedKMeters += lanes * distance / 1000.0
145         self.resetFitness()
146 
147     # remove a new nonregular track between two nodes
148     def removeAdditionalTrack(self, source, target, tid):
149         if source not in self.newTracks:
150             return False
151         if target not in self.newTracks[source]:
152             return False
153         trackR = None
154         for track in self.newTracks[source][target]:
155             if track["id"] == tid:
156                 trackR = track
157 
158         if trackR == None:
159             return False
160 
161         self.newInsertedKMeters -= trackR["lanes"] * trackR["distance"]
162         self.newTracks[source][target].remove(trackR)
163         self.resetFitness()
164 
165         return True
166 
167     def printDesign(self, outputFilename="output", outputFormat=None):
168         edgeColors = []
169         for track in self.problem.tracks:  # regular tracks
170             if (track["lanes"] + self.tracks[track["id"]]) == 0: continue
171             color = ""
172             if self.vehiclesByRoad[track["id"]] <= 7 * (track["distance"] /
     1000.0) * (
173                     track["lanes"] + self.tracks[track["id"]]):
```

```
174                 color = "green"
175             elif self.vehiclesByRoad[track["id"]] <= 11 * (track["distance
    "] / 1000.0) * (
176                     track["lanes"] + self.tracks[track["id"]]):
177                 color = "yellow"
178             elif self.vehiclesByRoad[track["id"]] <= 22 * (track["distance
    "] / 1000.0) * (
179                     track["lanes"] + self.tracks[track["id"]]):
180                 color = "orange"
181             elif self.vehiclesByRoad[track["id"]] <= 28 * (track["distance
    "] / 1000.0) * (
182                     track["lanes"] + self.tracks[track["id"]]):
183                 color = "purple"
184             else:
185                 color = "red"
186             edgeColors.append(color)
187         for s in self.newTracks:  # non-regular tracks
188             for t in self.newTracks[s]:
189                 for track in self.newTracks[s][t]:
190                     if track["lanes"] == 0: continue
191                     color = ""
192                     if self.vehiclesByRoad[track["id"]] <= 7 * (track["
    distance"] / 1000.0) * (track["lanes"]):
193                         color = "green"
194                     elif self.vehiclesByRoad[track["id"]] <= 11 * (track["
    distance"] / 1000.0) * (track["lanes"]):
195                         color = "yellow"
196                     elif self.vehiclesByRoad[track["id"]] <= 22 * (track["
    distance"] / 1000.0) * (track["lanes"]):
197                         color = "orange"
198                     elif self.vehiclesByRoad[track["id"]] <= 28 * (track["
    distance"] / 1000.0) * (track["lanes"]):
199                         color = "purple"
200                     else:
201                         color = "red"
202                     edgeColors.append(color)
203 
204         # creating graph for presentation
205         graph = nx.MultiDiGraph()
206 
207         i = 0
208         for node in self.problem.nodes:
209             graph.add_node(str(self.problem.nodes[node]["id"]))
210 
211         # arcs
212         for track in self.problem.tracks:  # regular tracks
213             if (track["lanes"] + self.tracks[track["id"]]) == 0: continue
```

```
214             graph.add_edge(track["s"], track["t"])
215         for s in self.newTracks:  # non-regular tracks
216             for t in self.newTracks[s]:
217                 for track in self.newTracks[s][t]:
218                     graph.add_edge(s, t)
219 
220         nodePos = {}
221         # for node in nodes:
222         for (node, data) in graph.nodes(data=True):
223             if node not in self.problem.nodes: continue
224             l = utm.from_latlon(self.problem.nodes[node]["lat"], self.
    problem.nodes[node]["long"])
225             x, y = l[0], l[1]
226             nodePos[node] = [x, y]
227 
228         # remove new nodes
229         nodosGraph = []
230 
231         for i in graph.nodes:
232             if i not in nodePos:
233                 nodosGraph.append(i)
234 
235         for j in nodosGraph:
236             graph.remove_node(j)
237 
238         # labeling arcs
239         edgeLabels = {}
240         for tid in self.tracks:  # regular
241             i = int(tid)
242             # don't show labels from new nodes
243             if i not in nodosGraph:
244                 lanes = self.tracks[tid]
245                 if lanes > 0:
246                     edgeLabels[(self.problem.tracks[i]["s"], self.problem.
    tracks[i]["t"])] = "+" + str(lanes)
247                 if lanes < 0:
248                     edgeLabels[(self.problem.tracks[i]["s"], self.problem.
    tracks[i]["t"])] = str(lanes)
249         for s in self.newTracks:  # non-regular tracks
250             for t in self.newTracks[s]:
251                 # dont show tracks from or to new nodes
252                 if s not in nodosGraph and t not in nodosGraph:
253                     for track in self.newTracks[s][t]:
254                         edgeLabels[(s, t)] = "New: (" + str(track["distance
    "]) + "m, " + str(
255                             track["maxspeed"]) + "km/h) x " + str(track["
    lanes"]) + "."
```

```
256
257         # nx.draw_networkx(graph, pos=nodePos,  arrows=True, arrowstyle
    ='-|>', with_labels=False, node_size=0, edge_color=edgeColors)
258         nx.draw(graph, pos=nodePos, arrows=True, arrowstyle='-|>',
    with_labels=False, node_size=0,
259                 edge_color=edgeColors)
260         nx.draw_networkx_edge_labels(graph, pos=nodePos, edge_labels=
    edgeLabels, font_color='black')
261         if outputFormat != None:
262             pyplot.savefig(outputFilename + "." + str(outputFormat), format
    =outputFormat)
263         # pyplot.savefig(outputFilename+".png", format="PNG")
264         # pyplot.savefig(outputFilename+".pdf", format="PDF")
265         pyplot.show()
```

utils.py

```
1 # returns zero from when dividing by zero
2 def indentZero(value, divisor):
3     if divisor != 0:
4         return value / divisor
5     else:
6         return 0
```

6.3 Artigo

OTIMIZAÇÃO DE MALHA VIÁRIA TERRESTRE ATRAVÉS DE ALGORITMO GENÉTICO

Gustavo Comiotto Schmitz, Rafael de Santiago

[1]Curso de Sistemas de Informação
Departamento de Informática e EstatísticaInstituto de Informática
Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brasil

gu_schmitz@hotmail.com, r.santiago@ufsc.br

Abstract. *According to [Vianna and Young 2015], the estimated total loss of GDP due to traffic in metropolitan regions is approximately 1.8% of GDP. Other factors can still be listed, such as damage to the health of those involved and an increase in the number of traffic accidents with victims, fatal or otherwise. In this context, this work aims to the optimization of the road network through an adaptation of the computational model of Salman-Alaswad. This model has been adapted to allow insertion and removal of roads and analyze the behavior in face of the inserted changes. To this end, the following actions had to done: A review of recent literature on optimization problems related to the traffic project; survey of problems similar to the one proposed in the research; proposition and development of the computational heuristic model. As a product of this work, a ranking was obtained with the best individuals per set of parameters found during the analysis phase, the set of parameters that presented the best results and the individual with the best "fitness" found in the experiments.*

Resumo. *Segundo [Vianna and Young 2015], a perda total estimada do PIB devido ao trânsito em regiões metropolitanas é de aproximadamente 1,8 % do PIB. Outros fatores ainda podem ser elencados, como danos a saúde dos envolvidos e um aumento no número de acidentes de trânsito com vítimas, fatais ou não. Nesse contexto, este trabalho tem como objetivo a otimização da malha viária através de uma adaptação do modelo computacional de Salman-Alaswad. Este modelo foi adaptado de modo a permitir a inserção e remoção de vias e analisar o comportamento frente as mudanças inseridas. Para isso, efetuou-se: revisão da literatura recente sobre problemas de otimização relacionadas ao projeto de trânsito; levantamento de problemas semelhantes ao proposto na pesquisa; proposição e desenvolvimento do modelo de heurística computacional. Como produto deste trabalho, obteve-se um ranking com os melhores indivíduos por conjunto de parâmetros encontrados durante a fase de análises, o conjunto de parâmetros que apresentou os melhores resultados e o indivíduo com melhor fitness encontrado nos experimentos.*

1. Introdução

Grandes engarrafamentos tornaram-se parte da paisagem das grandes metrópoles mundiais. Somadas ao aborrecimento causado pela situação, há também perdas econômicas

expressivas. A solução para o problema depende de uma melhor compreensão do uso e da ocupação do solo urbano [Maciel 2008]. Além do que foi mencionado acima, soma-se o fato de que a frota veicular terrestre brasileira aumentou substancialmente nos últimos anos. De acordo com [brasileiro de geografia e estatística IBGE 2021], o estado de Santa Catarina conta hoje com um pouco mais de 7,1 milhões de habitantes, no mesmo viés, segundo o [estadual de trânsito DETRAN 2021] a frota catarinense conta com quase 5,5 milhões de veículos. Fazendo uma conta muito simples, onde dividimos o número de veículos pelo total de habitantes percebemos que o estado possui aproximadamente 0,77 veículo por habitante. Tal dado ilustra uma série de problemas, dentre eles a ineficiência do transporte público existente e a falta de alternativas, de preferência sustentáveis, para enfrentar o trânsito, como a falta de ciclovias nas grandes regiões metropolitanas. Isso coloca uma pressão considerável sobre os governos locais e nacionais desenvolver uma infraestrutura de transporte para acompanhar a demanda cada vez maior da população por mobilidade e transporte de mercadorias [Salman and Alaswad 2018].

Um exemplo de cidade com problemas de tráfego terrestre é Florianópolis. Ela possui indicadores de qualidade de trânsito desfavoráveis, onde trajetos considerados curtos podem facilmente passar de uma hora de duração. Esses mesmos trajetos ainda podem ser mais duradouros quando imprevistos acontecem, como acidentes e chuvas torrenciais. Tal fato é ainda mais grave, quando se trata de uma cidade que possui o turismo como um dos seus pilares financeiros. Com isso, tanto o governo como o comércio, deixam de arrecadar recursos. Por parte do governo, os recursos poderiam ser revertidos em diversas áreas, inclusive na mobilidade urbana.

O Network Design Problem (NDP) visa encontrar o conjunto ideal de vias que devem ser melhorados em uma rede rodoviária, a fim de atingir um determinado objetivo (minimizar congestionamento, poluição ou consumo de energia) [Leblanc 1975, Poorzahedy and Turnquist 1982]. Esse problema é endereçado em vários trabalhos, os quais se utilizam de diversos algoritmos e métodos heurísticos.

O modelo NDP apresentado por [Salman and Alaswad 2018] propõe a redução dos congestionamentos utilizando como modelo computacional uma composição entre a metaheurística de cadeias de Markov e algoritmo genético. Onde otimiza o padrão de tráfego da rede, convertendo seletivamente estradas de mão dupla em estradas de mão única para chegar a um melhor desempenho da rede rodoviária (ou seja, congestionamento total da rede) sem a necessidade de realização de obras de infraestrutura, como construção de estradas ou adição de faixas [Salman and Alaswad 2018].

Com isso, este trabalho utiliza o problema dos congenstionamenos caracterizados como NDP utilizando o modelo de [Salman and Alaswad 2018] com o objetivo de otimizar o trânsito em diversas regiões, estendendo o modelo para lidar com inclusão e remoção de vias. Para isso, será utilizado a base de dados OpenStreetMap (openstreetmap.org), já o modelo computacional desenvolvido será uma adaptação do modelo de [Salman and Alaswad 2018]. Feito isso, com posse dos resultados, será efetuada uma análise dos dados obtidos através do modelo computacional desenvolvido.

2. Fundamentação Teórica

Segundo [Gallo et al. 2010], network design problems consiste em otimizar as características de uma rede viária urbana sem fornecer intervenções infraestruturais (projeto

de novas estradas, alargamento de estradas existentes, etc.); podendo envolver toda a rede rodoviária da cidade ou parte dela. Em geral, são considerados problemas de alta complexidade e por isso demandam muitos recursos, principalmente computacionais para sua resolução parcial ou total.

Para [Caggiani et al. 2017], O NDP é geralmente formulado como um problema de otimização de dois níveis para refletir os diferentes objetivos dos dois tomadores de decisão, que são os usuários da rede e o planejador. Os usuários da rede são livres para escolher suas rotas de forma que seus custos individuais de viagem sejam minimizados, considerando que o planejador visa fazer o melhor uso dos recursos limitados para otimizar o desempenho da rede (por exemplo, reduzindo o congestionamento, minimizando impacto ambiental e maximização de rendimentos), levando em conta o comportamento de escolha de rota dos usuários.

Modelo de usuário

Em um Network Design Problem, tradicionalmente é desenvolvido e exposto um modelo de usuário, que corresponde ao comportamento do mesmo dentro da rede. Seja para um elétron em uma rede elétrica ou para um veículo dentro de uma malha viária, é de suma importância entender o comportamento deles para poder prever e indicar possíveis alternativas. Segundo [Duell et al. 2016], os modelos de usuário são capazes de incorporar as diferentes opções de caminhos para cada usuário, podendo ser implementado por exemplo para estimar o tempo de viagem de um veículo, bem como seu consumo.

No modelo proposto por [Salman and Alaswad 2018], o comportamento dos usuários nas vias durante a execução do modelo é baseado na densidade máxima de veículos, ou seja, nos focos de congestionamentos. Diferentemente do comportamento ilustrado nos trabalhos de [Caggiani et al. 2017] e [Poorzahedy and M. Rouhani 2007], em que baseia-se no tempo de viagem entre dois pontos dentro da rede.

3. Trabalhos Relacionados

Os trabalhos aqui apresentados foram escolhidos após uma pesquisa por artigos relacionados a Road Network Design Problems filtrando pelos mais recentes na plataforma Scopus. Identificou-se 5 trabalhos, dentre eles removemos o primeiro por não ser mais do nosso interesse. Também removemos o segundo por envolver características muito específicas para o modelo genérico que propomos neste trabalho. Quanto ao terceiro, é um dos trabalhos que optamos descrevê-lo abaixo. Já os últimos dois trabalhos, houve a necessidade de descartá-los devido ao fato de não termos acesso aos mesmos de maneira gratuita.

3.1. Alleviating road network congestion: Traffic pattern optimization using Markov chain traffic assignment

Neste trabalho, os autores propõem uma solução para mitigar a reincidência dos congestionamentos nas vias rodoviárias através de um algoritmo genético com a utilização de cadeias de Markov para avaliação do modelo de ação. Tal otimização é efetuada ao inverter determinadas vias. Com isso, não são necessárias obras de infraestrutura, visto que está acontecendo uma alteração no sentido das faixas e não um acréscimo.

Para realização dessa otimização das vias, os autores optaram por utilizar o critério de densidade máxima de veículos numa estrada como principal indicador de desempenho da rede analisada. Para se mensurar a densidade, os autores utilizam a equação abaixo:

$$D_i = \frac{V\pi_i}{L_i N_i}, \tag{1}$$

na qual V representa o número total de veículos na rede rodoviária, π_i representa o vetor de distribuição estacionária, L_i representa o comprimento das estradas e N_i o número de faixas.

Os autores realizaram um experimento utilizando parte da cidade de Abu Dhabi, Emirados Árabes Unidos, onde a rede analisada contava com cerca de 360 estradas e aproximadamente 10.000 veículos transitando na rede. Com isso, eles obtiveram como resultado uma densidade máxima de veículos reduzida de 34,2 para 19,8 veículo/km/via, o que representa uma melhoria de 42%. Para se chegar a tal valor de melhoria foi necessária a inversão de sentido do fluxo de 24 estradas. A Figuras 1 e 2 ilustram os resultados da execução do modelo

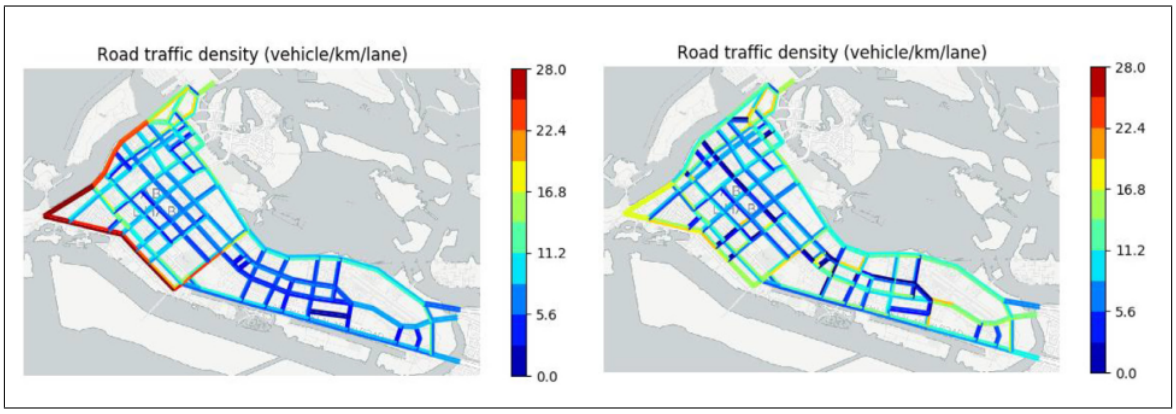


Figura 1. Densidade das vias antes e depois da execução do modelo. [Salman and Alaswad 2018]

Outras execuções ainda foram efetuadas pelos autores supondo outras realidades. Por exemplo, partindo do pressuposto que deseja-se alterar o mínimo de vias possível. Com isso, outros resultados foram obtidos, tanto na eficiência do modelo quanto nas ruas de fato alteradas, bem como sua quantidade. Outro exemplo é a aplicação do modelo em resposta a crises, como um incidente localizado. Novamente resultados totalmente diferentes foram encontrados.

Portanto, será ampliada a ideia [Salman and Alaswad 2018] para, não apenas inverter o sentido, mas inserir e remover vias no plano viário a ser analisado. De forma que, as novas soluções encontradas possam ser as mais adequadas para uma determinada região.

3.2. Método computacional para otimização do projeto da malha viária de Florianópolis-SC

Seguindo a mesma linha do trabalho anteriormente descrito, os autores também buscam mitigar problemas de congestionamentos em vias urbanas inspirado no trabalho

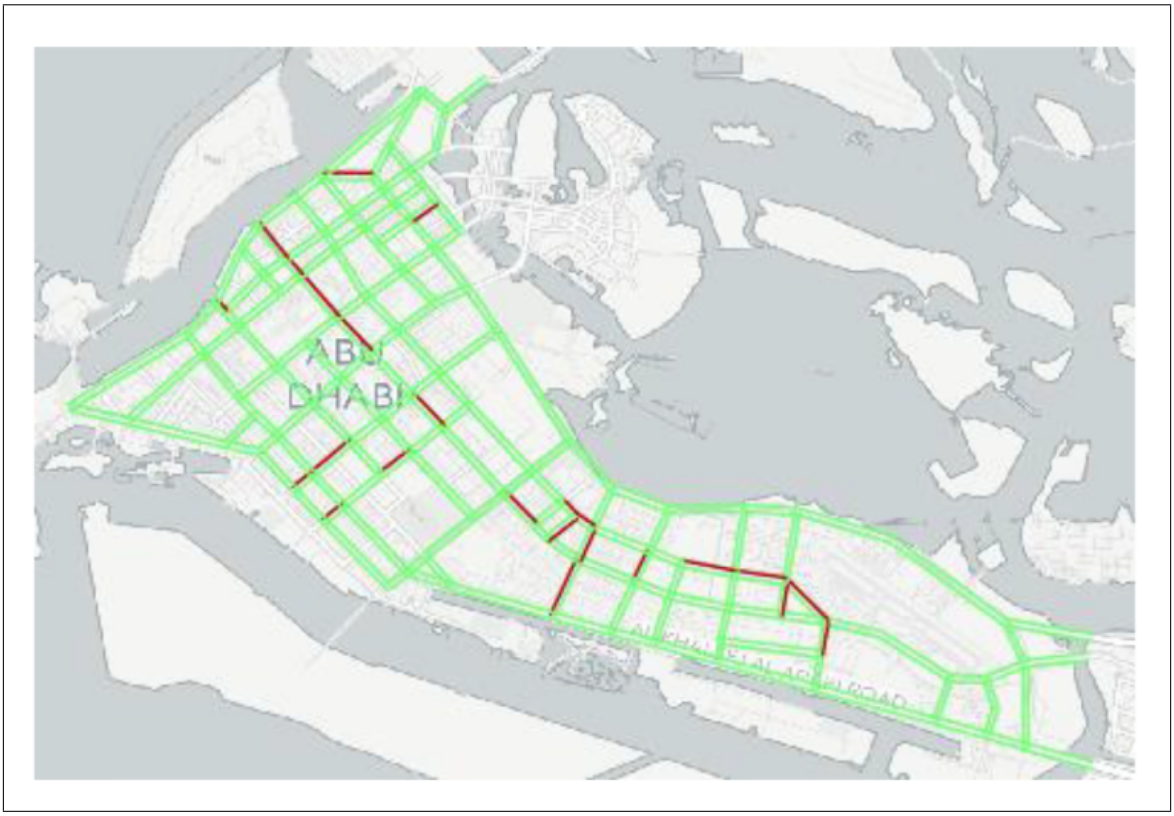


Figura 2. As 24 estradas com fluxo invertido. [Salman and Alaswad 2018]

de [Salman and Alaswad 2018], porém com cenário e contexto diferentes. Neste trabalho o problema a ser abordado tem como cenário a capital catarinense, Florianópolis [TOMASZEWSKI and de Santiago 2020].

Os autores subdividiram a pesquisa em 6 passos, sendo eles:

- 1 - Identificar as características e gargalos principais de malha viária de Florianópolis (parcial);
- 2 - Enumerar os principais métodos computacionais para o problema de planejamento da malha viária (total);
- 3 - Listar ranqueamento dos métodos computacionais mais adequados ao município de Florianópolis (total);
- 4 - Propor novo método computacional inspirado nos métodos ranqueados e aderentes à realidade do município de Florianópolis (total);
- 5 - Desenvolver método proposto (total);
- 6 - Avaliar o método proposto (total).

Os autores enfrentaram dificuldades na obtenção dos dados para pesquisa, desde uma grande burocracia por parte dos órgãos públicos de trânsito até o alto custo financeiro para obtenção dos mesmos por parte das empresas privadas. Diante desse cenário, optou-se por utilizar a plataforma OpenStreetMap (openStreetMap.org) por se tratar de uma ferramenta gratuita e com ampla documentação.

Quanto ao método proposto, os autores propuseram um algoritmo genético com funções capazes de calcular a probabilidade de um indivíduo passar do ponto a para o

ponto b e também calcular a densidade de cada arco do conjunto de arcos existentes. Para a obtenção dos resultados, foram utilizados dois dos principais operadores genéticos: Mutação e Crossover. Durante a execução da fase de mutação, o algoritmo seleciona aleatoriamente uma via de mão dupla e a torna de mão única. Já na fase de crossover, são selecionados aleatoriamente dois indivíduos para serem os pais e neles é selecionado um ponto p também de maneira aleatória. A partir de p são gerados os indivíduos filhos que darão prosseguimento ao processo.

Para avaliar o método, os autores criaram uma instância artificial com 16 vias, e deixando predeterminado o número de execuções (5), total de gerações em cada execução (50) e com 10 e 25 indivíduos.

Segundo os autores, tais dados ilustram uma estabilidade nos melhores indivíduos gerados a partir de determinada geração, para 25 indivíduos verificou-se a partir da sexta, já para 10 indivíduos a partir da nona. Tal fato comprova a influência da quantidade de indivíduos na determinação dos resultados.

O método foi desenvolvido na linguagem de programação Python e se encontra num repositório público no Github (https://github.com/Seis/NDP2020-Tomaszewski-Santiago).

Assim como descrito em 3.1, será ampliada a ideia de [TOMASZEWSKI and de Santiago 2020] para contemplar também a inserção e remoção de vias no plano viário a ser analisado para este trabalho.

3.3. A general methodology for reducing computing times of road network design algorithms

Neste trabalho os autores têm como proposta a redução do tempo computacional para um Road Network Design Problem (RNDP) voltado para otimização rodoviária para zona rural, mais especificamente para a cidade de Vilnius (Lituânia). Para essa realizar tal fato, a cada iteração do algoritmo de solução de rede, exceto o primeiro, os fluxos de tráfego de equilíbrio obtidos no final da iteração anterior são utilizados como iniciais no próximo procedimento de atribuição. Para os autores, a abordagem exposta só é viável se a topologia da rede rodoviária permanecer inalterada ao longo das iterações.

Os autores realizaram um experimento a fim de comparar a abordagem proposta com a tradicional, eles subdividiram a área do condado de Vilnius em 235 zonas de tráfego, onde a cidade de Vilnius foi subdividida utilizando uma grade de 2,5 km, já o restante da área com uma grade de 10,0 km, conforme a figura 3.

O próximo passo foi obter o modelo da rede rodoviária usando grafos a partir das características das estradas. O grafo presente no artigo possui um total de 5871 km de estradas, 590 nós e 250 centróides. Para o teste em questão, os autores utilizaram apenas as estradas principais, linhas verdes, e nacionais, linhas azuis, que representam 865 km e 1.713 km respectivamente.

Segundo os autores, os principais resultados da abordagem proposta é a redução do tempo computacional em 40,74% em uma análise anual, e tal valor pode dobrar no caso específico de uma análise do tráfego dos horários matutinos em dias úteis. Obtendo assim respostas mais rápidas e com um custo computacional consideravelmente mais baixo. Para os autores, essa redução só foi possível pelo fato de o algoritmo MSA

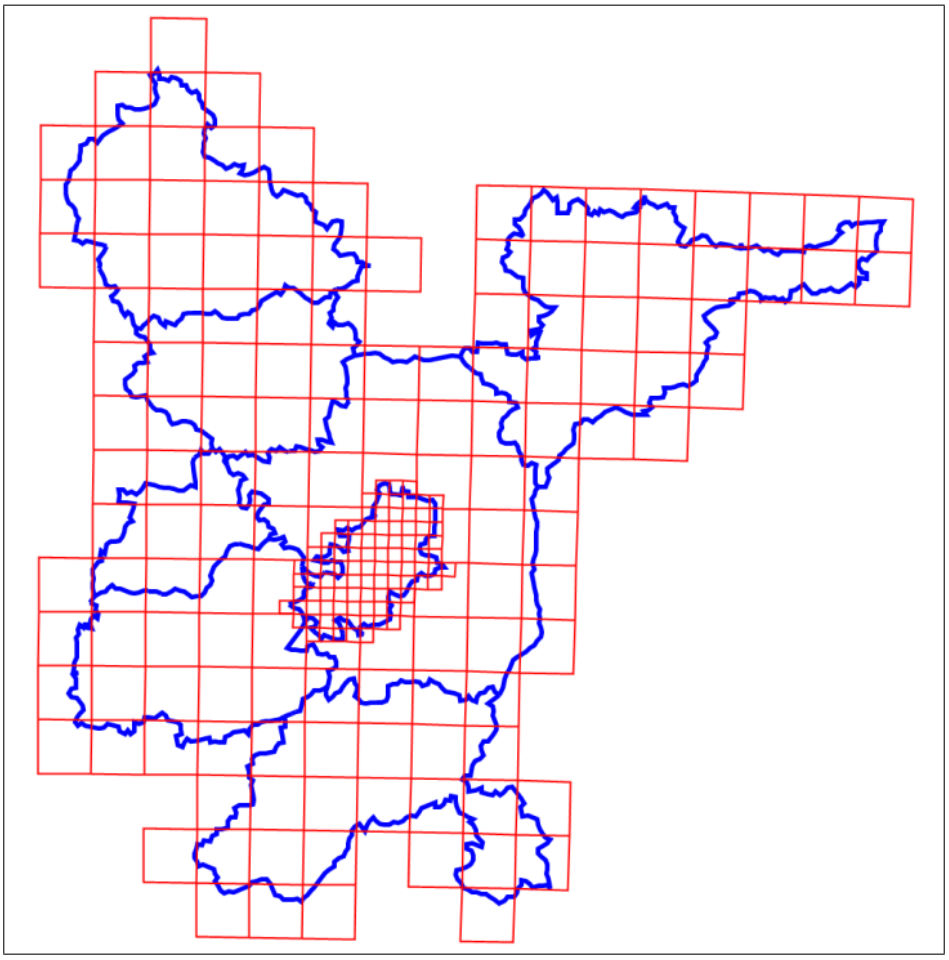


Figura 3. Zona do condado de Vilnius [Zilioniene et al. 2019]

utilizado adotar os fluxos iniciais de equilíbrio de tráfego baseado na iteração anterior do algoritmo de solução da rede, enquanto na abordagem tradicional, utiliza-se os fluxos iniciais assumindo valor igual a 0.

4. Especificação e Métodos

Neste capítulo, são especificados alguns pontos do trabalho como: apresentação de um algoritmo genético e seus operadores, a representação de um indivíduo, a população inicial, o critério de avaliação do modelo proposto e o plano de experimentação do mesmo.

4.1. Algoritmo genético

O algoritmo genético recebe como dados de entrada um conjunto de estradas e dados relacionados, tais como *travel time* e velocidade máxima de cada via. Através disso, o algoritmo genético inicia sua execução em busca do projeto de trânsito com o melhor *fitness* (inspirado no modelo de [Salman and Alaswad 2018], descrito na Seção 3.1).

Para os dados de entrada que serão obtidos a partir da ferramenta OpenStreetMap, as informações capturadas serão: a malha viária da região, com a velocidade máxima de cada via, a quantidade de pistas por via, direção de cada pista. Com esses dados, pretende-se conseguir representar o projeto de trânsito atual da cidade. A ideia é a de que o plano

da cidade seja utilizado como base para os indivíduos gerados na população inicial e nas demais etapas do algoritmo genético.

As subseções a seguir apresentam os aspectos considerados durante o projeto do algoritmo genético.

4.1.1. Representação de um indivíduo

Cada indivíduo (cromossomo) representará um projeto de trânsito da cidade, inspirado nas vias existentes da região a ser analisada. Cada segmento de uma via será considerado um gene, ou seja, uma parte do cromossomo. Cada gene possuirá os seguintes atributos: o sentido da via, a quantidade de faixas já existentes e dados de localização. Como o algoritmo genético considerará a possibilidade de criação de novas vias, gene novos podem ser adicionados a um cromossomo. Durante a execução do algoritmo, esses atributos são manipulados pelos operadores genéticos afim de aumentar o *fitness* do modelo.

4.1.2. Operadores genéticos

O modelo proposto neste trabalho utiliza os operadores genéticos: Seleção, Crossover e Mutação.

Seleção

Neste trabalho, foram implementadas três técnicas de seleção diferentes: "Torneio", "Roleta" e "Aleatória". A técnica de "Torneio" consistiu em selecionar aleatóriamente 5 dentre a geração atual de indivíduos, após esse passo, é selecionado para a próxima etapa o indivíduo que apresentar o melhor *fitness* entre os 5 selecionados previamente.

A técnica de "Roleta" consistiu em, primeiramente, somar todos os valores de *fitness* de todos os indivíduos. Tendo esse valor em mãos, é escolhido um valor aleatório entre 0 e o total da soma feita anteriormente. Então é realizada uma iteração entre todos os indivíduos da geração atual, incrementando o valor de uma variável com seu valor de *fitness*, e verificando se o valor dessa variável é maior que o valor aleatório escolhido anteriormente. Caso a condição descrita anteriormente seja verdadeira, o indivíduo é escolhido, caso contrário, ignora-se o indivíduo e a iteração é continuada.

Por último, a técnica "Aleatória" escolhe um indivíduo dentre todos os presentes na atual geração.

Crossover

O método de *crossover* desenvolvido para este trabalho consiste em analisar cada alteração, em relação ao plano viário original, dos indivíduos pais. Cada alteração dos pais tem 50% de chance de integrar indivíduo filho.

Mutação

A técnica de Mutação apresentada nesse trabalho consiste em, com base na taxa de mutação definida no início da execução do algoritmo, modificar aleatóriamente apenas as estradas pré-existentes no plano viário original. As modificações variam entre: inserção de uma nova faixa num trecho de estrada ou inserção de uma nova faixa em todos os trechos de uma estrada.

4.1.3. Fitness

O *fitness* no modelo proposto por este trabalho utiliza, diferentemente do modelo de [Salman and Alaswad 2018] que utiliza a densidade máxima de veículos numa estrada, o inverso da densidade média de veículos de todas as estradas do plano viário. A fórmula abaixo descreve de maneira detalhada a obtenção deste valor:

$$DM = \frac{DC \cdot 100}{DS},$$

onde DM representa a densidade média, DC representa o número de estradas do plano e DS o somatório das densidades de veículos de todas as estradas. O indivíduo que obtiver o maior resultado ao final deste cálculo será considerado o mais apto.

4.1.4. População inicial

A população inicial a ser utilizada pelo modelo desenvolvido nesse trabalho consiste num conjunto de N indivíduos, ou seja, N possíveis projetos de otimização da malha viária de uma determinada região.

Para definir a população inicial, os N indivíduos deverão conter mudanças aleatoriamente selecionadas a partir do plano atual da região de entrada do algoritmo (bairro, cidade, estado, . . .).

4.2. Delineamento do experimento

Para a execução dos experimentos foi utilizado um notebook com as seguintes características:

- Marca: Dell.
- Mémoria RAM: 32 GB.
- Memória: 2TB.
- Processador: Intel core i7 11th.
- Placa de vídeo: NVIDIA GP108M [GeForce MX330].

Com as características acima descritas, este notebook levou cerca de duas horas para executar as 72 combinações de conjuntos de parâmetros possíveis, sendo repetida 10 vezes para cada conjunto.

4.3. Roteiro de experimentos

Os experimentos efetuados neste trabalho são basicamente execuções do modelo com variações dos parâmetros existentes no algoritmo genético, com objetivo de detectar em quais gerações acontece convergência entre eles, considerando o tamanho da instância de entrada (número de trechos da via). Com isso, fica evidente a influência de cada parâmetro sobre o resultado da execução do modelo.

Para a execução dos testes, foi selecionado como plano viário base o município de Borá, localizado no interior do estado de São Paulo e sendo considerado o menor município do estado segundo a revista [Veja 2019] no quesito população. A escolha por uma cidade pequena se deu pela maior praticidade na execução dos experimentos de acordo com os recursos computacionais presentes no momento. Outros dois parâmetros foram deixados de maneira fixa durante todo o experimento, são eles: Número de gerações(50) e *budget* disponível para alteração na malha viária, sendo este de 10 quilômetros. A lista abaixo, ilustra os demais parâmetros que serão alterados durante a execução do algoritmo genético e seus valores, respectivamente:

- População inicial: 10, 15 e 20 Indivíduos.
- Taxa de mutação: 1% e 10%.
- Taxa de indivíduos elitistas por geração: 10% e 20%.
- Técnicas de seleção: Torneio, Roleta e Aleatória.
- Taxa de filhos gerados por geração: 25% e 50%.

Tomando tais parâmetros como base, será executado o algoritmo genético 10 vezes com cada uma das 72 combinações existentes.

5. Análise de Resultados: o que foi analisado

Feita a execução de todos os cenários descritos anteriormente em 4.3, são efetuadas as seguintes análises:

- Através de gráficos, definir uma tendência de comportamento do algoritmo, apresentando os conjuntos de parâmetros que obtiveram, em média, os melhores indivíduos.
- Através de gráficos e de uma tabela, ilustrar os indivíduos que obtiveram os melhores *fitness*, bem como as modificações efetuadas pelo algoritmo em cada um deles.
- Definição do melhor indivíduo encontrado dentre todas as execuções do algoritmo genético. Sendo este, o indivíduo que possuir o maior *fitness*.

5.1. Resultados e Análises

Esta seção se destina a exibir e comentar os resultados obtidos.

5.2. Melhores conjuntos de parâmetros

Com base nos dados obtidos, foi efetuado um cálculo de média, onde soma-se os valores de média de *fitness* por geração em cada execução, conforme ilustrado abaixo:

$$FMCP = \frac{\frac{SMFG_1}{NE} + \frac{SMFG_2}{NE} + ... + \frac{SMFG_G}{NE}}{G} \quad (2)$$

no qual, $FMCP$ indica o *fitness* médio por conjunto de parâmetros. O $SMFG_i$ representa o somatório das médias de *fitness* da i-ésima geração em todas as execuções do algoritmo com determinado conjunto de parâmetros. NE representa o número de execuções do algoritmo, já G representa o número de gerações pré-definidas. Essa equação é utilizada na Tabela 5.2.1.

Dado este cenário, obteve-se como resultado as 10 melhores médias de *fitness* por conjunto de parâmetros, conforme gráfico da Figura 4. Para o gráfico, é considerado "fitness" a equação abaixo:

$$FitnessPorGeracao = \frac{SMFG_i}{NE}. \tag{3}$$

no qual i corresponde a i-ésima geração.

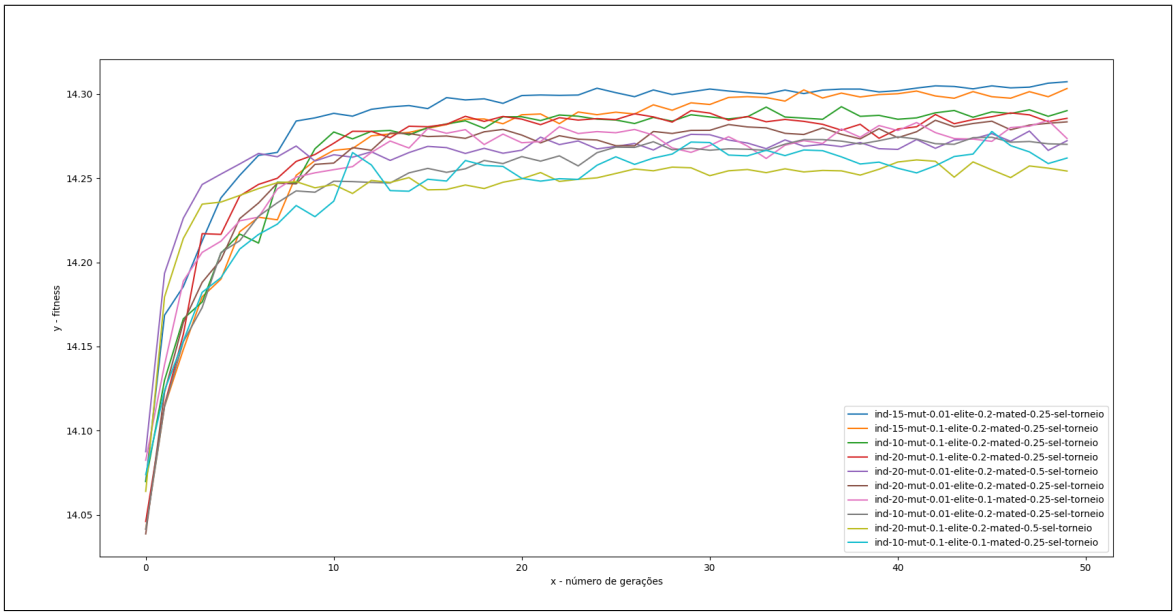


Figura 4. Os 10 melhores conjuntos de parâmetros com base no *fitness* médio por geração.

Analisando o gráfico da Figura 4, pode-se concluir que o algoritmo tende a apresentar os melhores indivíduos, em média, quando a técnica de seleção utilizada é a de "Torneio", visto que todos os registros presentes no gráfico utilizaram tal técnica. Há outros dois pontos a serem observados, o primeiro refere-se as taxas de indivíduos elitistas e de filhos gerados através da etapa de *crossover*. Neste primeiro ponto, percebe-se que tais parâmetros são importantes na obtenção do resultados, mas de maneiras opostas. A taxa de elitismo se comportou de maneira progressiva, ou seja, quanto maior, melhor o resultado. Com relação ao parâmetro taxa de filhos gerados através da etapa de *crossover*, observa-se que quanto maior o número de filhos, menor a qualidade esperada por geração. Já o segundo ponto, refere-se a taxa de mutação e o tamanho da população, tais fatores não se mostraram relevantes, pois seus resultados são equilibrados. Portanto, após os experimentos verificou-se que, em 50 gerações e com 10 execuções do algoritmo com cada conjunto de parâmetros, os melhores resultados obtidos possuíam os seguintes parâmetros:

- População Inicial: 15 indivíduos.
- Taxa de mutação: 1%.
- Taxa de indivíduos elitistas por geração: 20%.
- Taxa de filhos gerados por geração: 25%.
- Técnica de seleção: Torneio.

5.2.1. Melhores indivíduos encontrados por conjunto de parâmetros

Para encontrar os melhores indivíduos por conjunto de parâmetros individualmente, foi feita uma análise pelo *fitness* do melhor indivíduo de cada execução. Dado esse cenário, gerou-se uma classificação com os melhores indivíduos.

Colocação	Conj. parâmetros	Fitness
1º	Indivíduos - 15 Tx. Mutação - 10% Tx. Elitismo - 20% Tx. Filhos gerados - 50% Téc. Seleção - Torneio	14.4661
2º	Indivíduos - 20 Tx. Mutação - 10% Tx. Elitismo - 20% Tx. Filhos gerados - 50% Téc. Seleção - Torneio	14.4102
3º	Indivíduos - 15 Tx. Mutação - 10% Tx. Elitismo - 10% Tx. Filhos gerados - 25% Téc. Seleção - Torneio	14.4015
4º	Indivíduos - 10 Tx. Mutação - 10% Tx. Elitismo - 10% Tx. Filhos gerados - 50% Téc. Seleção - Torneio	14.4005
5º	Indivíduos - 15 Tx. Mutação - 1% Tx. Elitismo - 10% Tx. Filhos gerados - 50% Téc. Seleção - Torneio	14.3926
6º	Indivíduos - 20 Tx. Mutação - 10% Tx. Elitismo - 10% Tx. Filhos gerados - 25% Téc. Seleção - Torneio	14.3863

Colocação	Conj. parâmetros	Fitness
7º	Indivíduos - 15 Tx. Mutação - 1% Tx. Elitismo - 20% Tx. Filhos gerados - 50% Téc. Seleção - Torneio	14.3862
8º	Indivíduos - 20 Tx. Mutação - 10% Tx. Elitismo - 20% Tx. Filhos gerados - 25% Téc. Seleção - Torneio	14.3794
9º	Indivíduos - 20 Tx. Mutação - 1% Tx. Elitismo - 10% Tx. Filhos gerados - 25% Téc. Seleção - Torneio	14.3745
10º	Indivíduos - 15 Tx. Mutação - 10% Tx. Elitismo - 10% Tx. Filhos gerados - 50% Téc. Seleção - Torneio	14.3745

Tabela 1. Os 10 melhores indivíduos encontrados por conjunto de parâmetros

Ao observarmos a tabela 5.2.1, é possível verificar as diferentes alterações efetuadas pelo algoritmo genético no plano viário da cidade de Borá-SP. As indicações que contenham um "+" acompanhado de um determinado valor N, indicam que houve um acréscimo de N faixas de trânsito naquele trecho de via. Já a outra indicação, sugere que houve a inserção de uma nova estrada no plano viário preexistente, de tamanho, velocidade máxima permitida e número de faixas também explicitados. As cores nos mapas indicam a intensidade do transito nas vias, quanto mais quente for a cor, mais intenso o trânsito. A Figura 5 mostra a evolução do *fitness* dos indivíduos classificados no decorrer das 50 gerações.

Com base nas informações do gráfico da Figura 5, percebe-se que novamente a técnica de seleção que apresentou os melhores resultados, assim como na seção 5.2, é a de "Torneiro". Portanto, pode-se concluir que o parâmetro mais influente nos resultados foi a técnica de seleção empregada, já os demais parâmetros demonstram-se equilibrados durante os experimentos. Outro ponto a ser destacado é o fato de nenhum dos registros do gráfico apresentar declínio durante o decorrer das gerações, isso se deu pelo fato do algoritmo possuir um caráter elitista. Isso garante que ao final da execução, teremos sempre o melhor indivíduo dentre todos os gerados.

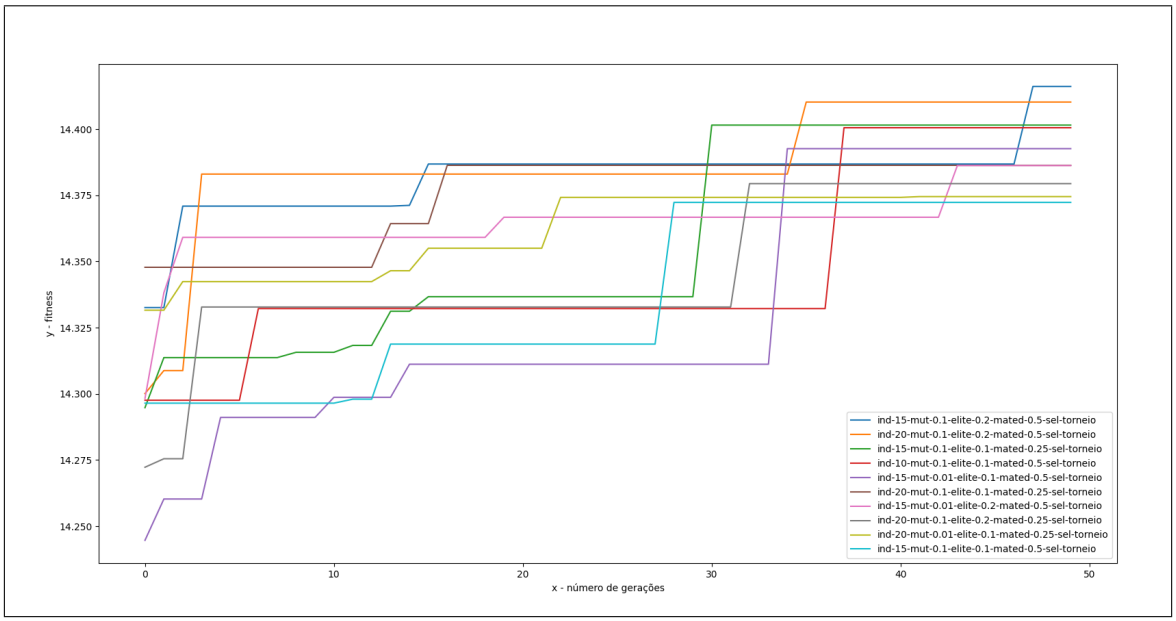


Figura 5. Os 10 melhores indivíduos com base no *fitness* por conjunto de parâmetros.

5.2.2. Melhor indivíduo encontrado

Após a execução dos experimentos com base no plano viário apresentado na Figura 6, o indivíduo que apresentou o melhor *fitness* com base nos parâmetros, fixos e variáveis, foi o da Figura 7.

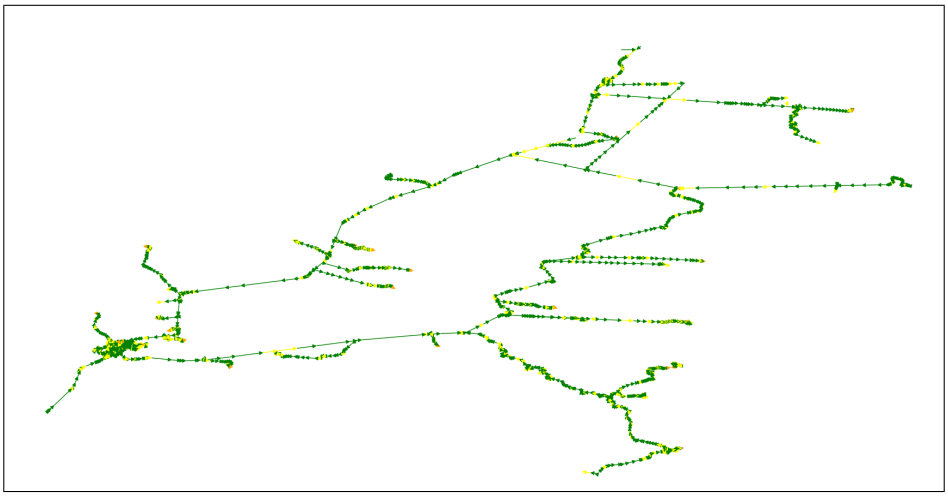


Figura 6

Comparado com o plano viário inicial do experimento, o indivíduo selecionado apresentou cerca de 27 faixas novas em trechos de estradas preexistentes e a inserção de uma nova pista de 2,38KM no plano viário da cidade analisada. Além disso, possui um *fitness* no valor de 14.4161, sendo levemente superior aos demais indivíduos. O conjunto de parâmetros utilizado para encontrar este indivíduo possui:

- Budget: 10 quilômetros.

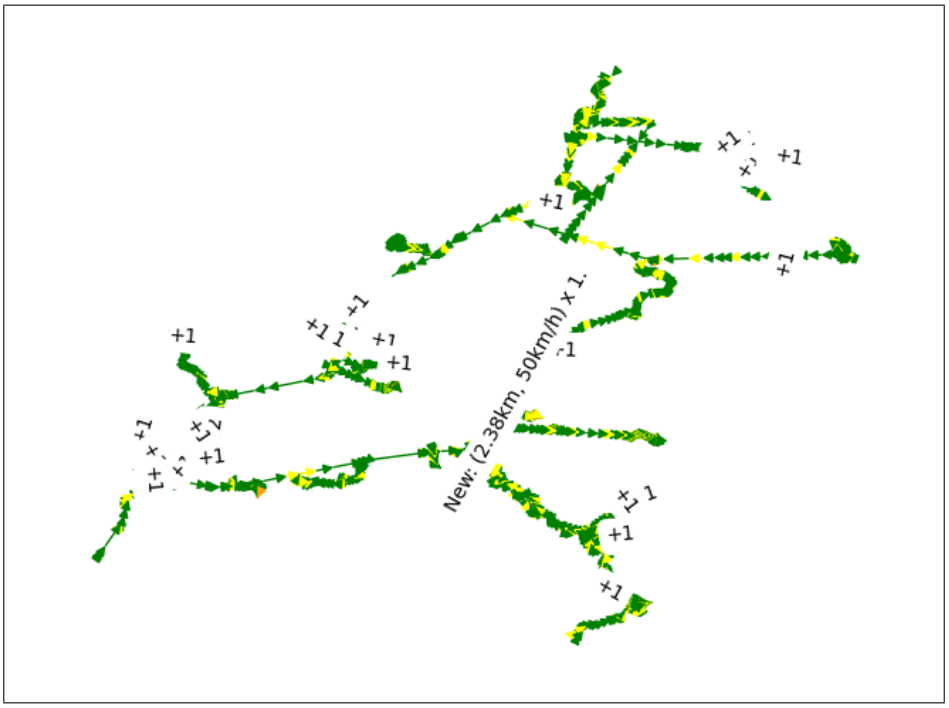


Figura 7. O melhor indivíduo encontrado durante os experimentos.

- Número de gerações: 50 gerações.
- População Inicial: 15 indivíduos.
- Taxa de mutação: 10%.
- Taxa de indivíduos elitistas por geração: 20%.
- Taxa de filhos gerados por geração: 50%.
- Técnica de seleção: Torneio.

6. Conclusões

Com o objetivo de otimizar a malha viária de uma determidada localidade reduzindo o número de congestionamentos, a abordagem trazida por este trabalho contempla a utilização de conceitos de computação evolutiva, mais especificamente a implementação de um algoritmo genético para tal tarefa.

Este trabalho apresenta um algoritmo genético capaz de modificar o plano viário de uma localidade, seja um bairro ou uma cidade, de forma a melhorar o trânsito. Para atingir esse objetivo, foi necessário efetuar um levantamento de trabalhos relacionados ao tema na literatura acadêmica, definir de uma localidade a ser analisada e o desenvolver um algoritmo genético adequado.

Após a realização dos experimentos descritos em 5.1, encontrou-se:

- o melhor plano viário, encontrado durante os experimentos, para a cidade de Borá-SP sendo o indivíduo representado na Figura 7, que possui um *fitness* de 14.4661.
- o conjunto de parâmetros que apresentou os melhores resultados, descrito em 5.2.
- o melhor indivíduo de cada combinação de parâmetros encontrado durante os experimentos.

Todos os objetivos definidos para este trabalho foram atingidos, porém sabe-se que as soluções encontradas não são as melhores possíveis, mas sim as melhores encontradas dentros dos parâmetros existentes e dos experimentos efetuados.

Trabalhos futuros

Abaixo são apresentados possíveis temas para trabalhos futuros:

- adaptar o algoritmo para levar em consideração a análise de *travel time* (tempo de viagem) e não mais a densidade média de veículos;
- expandir o número de cidades analisadas;
- incorporar os recursos financeiros ao budget, para contemplar não apenas a quilometragem disponível;
- incorporar a análise de relevo ao algoritmo, para que não haja inserção de vias onde financeiramente não é possível.

Referências

brasileiro de geografia e estatística IBGE, I. (2021). Estatísticas gerais.

Caggiani, L., Camporeale, R., and Ottomanelli, M. (2017). Facing equity in transportation network design problem: A flexible constraints based model. *Transport Policy*, 55:9–17.

Duell, M., Gardner, L., and Waller, S. (2016). Policy implications of incorporating distance constrained electric vehicles into the traffic network design problem. *Transportation Letters*, 10:1–15.

estadual de trânsito DETRAN, D. (2021). Veículos em circulação em santa catarina.

Gallo, M., D'Acierno, L., and Montella, B. (2010). A meta-heuristic approach for solving the urban network design problem. *European Journal of Operational Research*, 201:144–157.

Leblanc, L. (1975). An algorithm for the discrete network design problem. *Transportation Science*, 9:183–199.

Maciel, V. (2008). Congestionamentos urbanos. *GV-executivo*, 7:20.

Poorzahedy, H. and M. Rouhani, O. (2007). Hybrid meta-heuristic algorithms for solving network design problem. *European Journal of Operational Research*, 182:578–596.

Poorzahedy, H. and Turnquist, M. (1982). Approximate algorithms for the discrete network design problem. *Transportation Research Part B: Methodological*, 16:45–55.

Salman, S. and Alaswad, S. (2018). Alleviating road network congestion: Traffic pattern optimization using markov chain traffic assignment. *Computers & Operations Research*, 99.

TOMASZEWSKI, M. and de Santiago, R. (2020). Método computacional para a otimização do projeto da malha viária de florianópolis-sc.

Veja (2019). Menor município do estado, borá teme junção com cidade maior.

Vianna, G. and Young, C. E. (2015). Em busca do tempo perdido: Uma estimativa do produto perdido em trÂnsito no brasil. *Revista de Economia Contemporânea*, 19:403–416.

Zilioniene, D., D'Acierno, L., Botte, M., and Gallo, M. (2019). A general methodology for reducing computing times of road network design algorithms. *International Journal of Supply and Operations Management*, 6:126–141.