Universidade Federal de Minas Gerais
Programa de Pós-Graduação em Engenharia Elétrica

Otimização da Programação Semafórica com Base em Modelos Matemáticos

Eric Wilian Lage Gonzaga

Dissertação de mestrado submetida à Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Minas Gerais, como requisito para obtenção do título de Mestre em Engenharia Elétrica.

Orientador: Prof. Dr. Eduardo Gontijo Carrano

Coorientador: Prof. Dr. Paulo Eduardo Maciel de Almeida

Belo Horizonte, 2016

Resumo

Nas últimas décadas, as médias e grandes cidades passaram a conviver com sérios problemas de gerenciamento do tráfego e congestionamentos em suas vias urbanas. Esses problemas tornaram a expansão da malha viária uma necessidade iminente. Porém, isso nem sempre é viável, seja por falta de espaço ou pelo alto custo. Portanto, uma engenharia de tráfego que dê soluções práticas, rápidas e eficientes se torna essencial. Dentre as estratégias possíveis, a otimização da programação semafórica se mostra interessante, visto seu potencial e baixo custo. Diversos trabalhos buscam otimizar a programação semafórica de forma a maximizar a velocidade média dos veículos nos trechos. Contudo, muitos deles utilizam simuladores de tráfego para avaliar os indivíduos gerados pelos algoritmos de otimização, o que se tornou um gargalo para o processo de otimização. Devido ao seu alto tempo de execução, esses simuladores impedem a otimização da programação semafórica em tempo real. Além disso, as programações obtidas eram avaliadas para apenas um cenário de fluxos de veículos no trecho, o que trazia dúvidas a respeito de seu funcionamento no mundo real, onde os fluxos podem variar ao longo do dia. Tendo em vista essas limitações, o objetivo deste trabalho é propor duas novas arquiteturas, que utilizam de modelos matemáticos determinísticos, a fim de obter programações semafóricas otimizadas e robustas em tempo real que minimizem os tempos gastos pelos veículos para percorrerem determinados trechos. Assim, foram realizados testes com o uso de algoritmos de otimização mono e multiobjetivo para trechos reais e os mesmos indicaram o sucesso das arquiteturas para esse objetivo.

Abstract

Nowadays large cities deal with serious problems in traffic management. Those problems demand urgent expansion of the road network. However, usually it is not feasible due to the lack of space or due to high costs involved. Therefore, a traffic engineering that provides fast and efficient solutions becomes essential. Among the possible strategies, the traffic light programming optimization arises as an interesting choice, since it is an effective and low-cost solution. Several studies propose to optimize the traffic light programming by maximizing the average speed of vehicles on road networks. However, many of these works use traffic simulators to evaluate candidate solutions, which can become a problem to the optimization process. These simulators require high runtime, which prevents the optimization of traffic light time in real time. Furthermore, the obtained solutions are evaluated for a single vehicle flow scenario, which causes doubts about its behaviour in the real world, in which vehicle flow can varies significantly along the day and from one day to another. Because of these limitations, the aim of this study is to propose two new architectures, which use deterministic mathematical models, in order to obtain robust and optimized traffic light programming in real time. These approaches aim to minimize the time spent by vehicles to travel on network. Experiments were performed using mono and multi-objective optimization algorithms for real scenarios and they indicated the success of such architectures for this purpose.

Agradecimentos

Gostaria de agradecer primeiramente ao meu Senhor Jesus Cristo, por ter me dado a vida, a salvação e por sempre cuidar de mim.

Agradeço a minha mãe Adriana, ao meu pai Wilian e ao demais familiares, que sempre me amaram, me apoiaram e tanto fizeram para que eu pudesse chegar até aqui.

Agradeço a minha namorada e aos meus amigos por estarem comigo e tornarem minha vida melhor.

Agradeço ao meu orientador Eduardo Carrano que me deu esta oportunidade e me suportou nos últimos dois anos e me ajudou sempre que foi necessário. Também ao meu coorientador Paulo Almeida por ter me indicado a área e me auxiliado nas dificuldades.

Por fim, agradeço aos demais professores que participaram da minha formação acadêmica e profissional até hoje.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de Acrônimos

AE: Algoritmos Evolucionários

AG: Algoritmo Genético

BHTRANS: Empresa de Transporte e Trânsito de Belo Horizonte S/A

CDA: *Crowding Distance Assignment*

CET-SP: Companhia de Engenharia de Tráfego de São Paulo

DE: *Differential Evolution*

DENATRAN: Departamento Nacional de Trânsito;

FNS: *Fast Nondominated Sorting*

GISSIM: *GIS Traffic Simulation*

GISSIM-TL: *GISSIM - Traffic Lights*

GPU: *Graphics Processing Units*

NSGA: *Nondominated Sorting Genetic Algorithm*

PSO: *Particle Swarm Optimization*

RWS: *Roulette Wheel Selection*

SUMO: *Simulation of Urban Mobility*

SUS: *Stochastic Universal Sampling*

VNS: *Variable Neighborhood Search*

Capítulo 1

Introdução

Nos últimos 20 anos, a melhoria do padrão de vida dos brasileiros e incentivos do governo, tais como a diminuição dos impostos sobre a venda de veículos, facilitaram a compra de veículos automotores (Downie, 2008). Consequentemente, houve um aumento na frota de veículos do país e as médias e grandes cidades passaram a conviver com sérios problemas de gerenciamento do tráfego e congestionamentos em suas vias urbanas, especialmente nos horários de pico.

Para demonstrar esses problemas, um relatório divulgado pela Companhia de Engenharia de Tráfego de São Paulo (CET-SP) (CET-SP, 2010) indicou que o fluxo de veículos da cidade de São Paulo cresceu mais de 40% entre anos de 1997 e 2009. Com isso, a velocidade média dos veículos no horário de pico diminuiu cerca de 33%, caindo de 17,5 km/h para 11,7 km/h. Vale destacar que a população da cidade teve um acréscimo de apenas 8% nesse período.

Em Belo Horizonte, o relatório de Informações da Mobilidade Urbana (BHTRANS, 2011), fornecido pela Empresa de Transporte e Trânsito de Belo Horizonte (BHTRANS), apontou o problema de aumento da frota de veículos como sendo ainda maior. Entre 1999 e 2010, o número de automóveis licenciados, que era de 491.332, passou a 937.819, ou seja, um aumento de 91%. Enquanto isso, o número de motocicletas cresceu 266%, passando de 44.634 para 163.489 no mesmo período. Assim, o Relatório de Evolução da Frota de Automóveis e Motos no Brasil (INCT, 2013) constatou que Belo Horizonte foi a metrópole brasileira que registrou o maior crescimento percentual no número de veículos automotivos no período entre 2001 e 2012. A frota total na metrópole que era de 841.060 veículos automotivos passou para 1.880.608, ou seja, um aumento de 123,6%.

No âmbito nacional, havia em 2001 aproximadamente 34,9 milhões de veículos, enquanto no ano de 2012 a frota total ultrapassou a marca de 76,1 milhões (INCT, 2013) (incremento de aproximadamente 118%). Enquanto isso, o crescimento populacional no Brasil, entre os dois últimos censos demográficos (2000 e 2010), foi de 12,3% (IBGE, 2000, 2010).

Dessa forma, os problemas de tráfego se agravam a cada dia. O crescente número de veículos e pedestres reforça a necessidade de expansão da malha viária. Porém, muitas vezes isso é inviável, seja por falta de espaço ou pelo alto custo. Portanto, uma engenharia de tráfego que dê soluções práticas, rápidas e eficientes se torna essencial. Para tal, os projetistas devem utilizar ferramentas e estratégias que sejam capazes de diminuir o impacto do crescimento da frota de veículos e, desse modo, garantir fluidez e segurança ao tráfego.

Dentre as estratégias apontadas, a otimização da programação semafórica tem sido uma das mais importantes (Costa, 2012) e já é aplicada por diversas empresas de tráfego urbano (Vilanova, 2005). Ela é fundamental para gerenciar movimentos conflitantes em cruzamentos e bifurcações, de forma a otimizar o tráfego e diminuir o tamanho das filas.

Os primeiros estudos sobre programação semafórica utilizavam métodos matemáticos determinísticos, como o proposto por Webster (1958), para otimizar os tempos de ciclos em interseções isoladas. Essa abordagem foi adotada em manuais semafóricos de todo o mundo, tais como DENATRAN (1984), TRB (2000) e CONTRAN (2012). Entretanto, por serem falhos na aplicação em redes de semáforos (Allsop and Charlesworth, 1977), que é a situação predominante no mundo real, perderam espaço para os simuladores de tráfegos, que eram capazes de avaliar os trechos como um todo. Além disso, os algoritmos evolucionários também trouxeram novas possibilidades de tratamento do problema (Sheffi and Powell, 1983; Saka et al., 1986).

Nesse contexto, Oliveira (2009) buscou maximizar a velocidade média dos veículos. Para tal, foi criado uma ferramenta capaz de gerar tempos semafóricos utilizando técnicas de inteligência computacional, acoplada ao simulador de tráfego OpenJUMP (2014). Costa (2012), por sua vez, estendeu o trabalho de Oliveira (2009) e modelou o problema com conceitos multiobjetivo. Seu intuito era maximizar a velocidade média dos veículos e minimizar a sua variância, de forma que as velocidades dos veículos fossem equilibradas em todo o trecho.

Contudo, o simulador de tráfego utilizado para avaliar as programações candidatas geradas pelos algoritmos evolucionários se tornou um gargalo. Devido ao seu uso, os trabalhos de Oliveira (2009) e Costa (2012) gastavam horas, ou até mesmo dias, para retornar boas soluções.

Por isso, era inviável realizar a otimização da programação semafórica em tempo real. Além disso, as programações obtidas eram avaliadas para apenas um cenário de fluxos de veículos nos movimentos, o que trazia dúvidas a respeito de seu funcionamento no mundo real, onde os dados de fluxos no trecho podem conter erros e variar ao longo do dia.

Tendo em vista essas limitações, o objetivo deste trabalho é propor duas novas arquiteturas, que voltam ao uso de modelos matemáticos determinísticos, a fim de se obter programações semafóricas otimizadas e robustas em tempo real, que minimizem os tempos gastos pelos veículos para percorrerem determinados trechos. Por fim, as arquiteturas são testadas em cenários reais, com o uso de algoritmos de otimização mono e multiobjetivo.

1.1 Estado da Arte

Nesta seção são apresentados alguns trabalhos da literatura que lidam com programação semafórica. Estes foram separados pelo modo com que abordam o problema.

1.1.1 Primeiras Abordagens

Os primeiros estudos sobre programação semafórica, datados da década de 1950, utilizavam métodos matemáticos determinísticos para calcular tempos de ciclos ótimos em interseções isoladas. Nesse cenário, Webster (1958) foi um dos pioneiros ao propor o Método do Grau de Saturação Máximo para calcular tempos ótimos de ciclo. Essa abordagem foi adotada em diversos manuais semafóricos, tais como (TRB, 1965, 2000, 2010; CONTRAN, 2012), que servem como base para definir as diretrizes em cidades de todo o mundo. A técnica também foi objeto de estudo em vários trabalhos, como (Sheffi and Powell, 1983; Vilanova, 2005; Guberinic et al., 2007; Kesur, 2009; Cervantes et al., 2009).

Nesse contexto, outra vertente de estudos foi iniciada a partir do Método de Cálculo do Tempo de Viagem, proposto por Davidson (1966, 1978), que também utilizava o grau de saturação máximo. Porém, esse método foi alvo de críticas devido a dificuldade de calibração de seus parâmetros, visto que não tinham conceitos bem definidos (Golding, 1977; Rose et al., 1989). Com isso, surgiu um grupo de pesquisas destinado a refinar esse método e resolver seus problemas. Esse grupo propôs, então, métodos alternativos para cálculo do tempo de viagem, sendo que os principais podem ser vistos nos trabalhos de Akçelik (1981, 1991) e de Tisato

(1991). Por fim, os novos métodos foram submetidos a testes com cenários reais para avaliar suas eficácias.

Os modelos matemáticos determinísticos trouxeram grandes contribuições para trabalhos práticos e teóricos sobre programação semafórica ao longo dos anos. No entanto, estes modelos possuíam limitações. Um exemplo disso, era a incapacidade de lidar com a variação dos fluxos de veículos ao longo dos tempos de verde (Webster, 1958). Além disso, eles tratavam apenas interseções isoladas e eram inviáveis para aplicações em redes de semáforos, situação que predomina no mundo real, onde deve-se ajustar um conjunto de ciclos simultaneamente (Allsop and Charlesworth, 1977).

1.1.2 Métodos Computacionais

Devido às limitações dos modelos matemáticos determinísticos e o avanço da computação, foram propostas novas abordagens para o problema de otimização da programação semafórica.

Robertson (1969) criou o *software* TRANSYT, que tinha como objetivo otimizar a alocação de tempos de verde em cada ciclo e o tempo de início dos ciclos em cada interseção (defasagem). Assim, o sistema é capaz de coordenar os semáforos presentes em um trecho. A otimização, por sua vez, era realizada por meio do método *hill-climbing* e a função objetivo utilizada foi uma combinação linear do atraso e do número de paradas.

Allsop and Charlesworth (1977) combinaram técnicas de cálculo dos tempos semafóricos e de balanceamento do tráfego em trechos. Dessa forma, esse trabalho teve caráter multiobjetivo, onde buscava-se a diminuição do atraso e o aumento da segurança dos veículos. Porém, só foi possível obter bons resultados em redes de pequeno porte.

Sheffi and Powell (1983) desenvolveram um trabalho cujo problema era encontrar os tempos de verde de todas as interseções para minimizar o tempo de viagem, considerando critérios de equilíbrio do trecho definidos pelo usuário. Para tal, foi utilizado um algoritmo de busca baseado em vetor gradiente. Contudo, esse método também só era viável para pequenas redes.

Outras abordagens para o problema de otimização semafórica podem ser citadas:

Multi-agentes ou Teoria de Jogos: (Bazan, 1995; Porche and Lafortune, 1998; Roozemond, 2001; Choy et al., 2003; Kosonen, 2003; Bazzan, 2005; de Oliveira, 2005; Cheng et al., 2006; Srinivasan and Choy, 2006; Alvarez et al., 2007; Lämmer and Helbing, 2008; Shamshirband et al., 2008; Xie et al., 2011);

Redes Neurais Artificiais: (Spall and Chin, 1997; Henry et al., 1998; López et al., 1999; Saito and Fan, 2000; Bingham, 2001; Guojiang and Youxian, 2005; Choy et al., 2006; Srinivasan et al., 2006; Shen and Kong, 2009);

Lógica *Fuzzy*: (Chiu and Chand, 1993; Kim, 1997; Hong et al., 1999; Lee and Lee-Kwang, 1999; Trabia et al., 1999; Chou and Teng, 2002; Murat and Gedizlioglu, 2005; Zarandi and Rezapour, 2009; Azimirad et al., 2010; Gokulan and Srinivasan, 2010; Ma et al., 2012);

***Branch-and-Bound*:** (Pillai et al., 1998; Yan et al., 2008);

***Swarm Intelligence*:** (de Oliveira and Bazzan, 2006; Renfrew and Yu, 2009).

Além desses trabalhos, surgiram também diversos simuladores microscópicos, como por exemplo DRACULA (ITS, 1993), OpenJUMP (2014), AIMSUN (TSS, 2015) e SUMO (ITS, 2015). Esses simuladores possuem um alto nível de detalhamento e avaliam cada um dos veículos, assim como outros componentes do sistema, de forma individualizada. A partir dessas ferramentas é possível, então, reproduzir o comportamento do tráfego. Seus resultados permitem a avaliação de várias situações reais de trânsito, assim como o cálculo do atraso dos veículos no trecho sob análise. No entanto, devido ao nível de detalhamento, essa é uma abordagem com alto custo computacional e requer um complexo processo de calibração.

1.1.3 Algoritmos Evolucionários

O advento dos algoritmos evolucionários também trouxe grandes contribuições para o tratamento do problema da programação semafórica. Eles têm sido utilizados desde a década de 1990 para programar semáforos em interseções isoladas ou redes de semáforos. Alguns exemplos desses trabalhos são (Foy et al., 1992; Hadi and Wallace, 1993; Oda et al., 1996; Park, 1998; Rouphail et al., 2000; Masterton and Topiwala, 2008; Sánchez et al., 2008; Turky et al., 2009).

Dentre os trabalhos, Vogel et al. (2000) realizaram a otimização dos tempos de verde utilizando um Algoritmo Genético (AG). Seu processo de otimização teve como base dados de fluxos de veículos locais e interseções isoladas. Assim, o objetivo desse trabalho era que os semáforos presentes em um ciclo pudessem sofrer adaptações de modo reativo em relação ao fluxo veículos em determinados horários. O Algoritmo Colônia de Formigas também foi utilizado em trabalhos similares, tais como (Zechman et al., 2010), (D'Acierno et al., 2010), (He and Hou, 2012) e (Putha et al., 2012).

Mais recentemente, Neto (2009) desenvolveu uma ferramenta de código aberto chamada GISSIM, incorporada ao OpenJUMP (2014), que realiza simulação microscópica de trânsito. A partir dessa ferramenta, Oliveira (2009) criou o GISSIM-TL, que possibilitava a obtenção de tempos semafóricos por meio de técnicas de inteligência computacional. Esse trabalho foi testado com um AG e um algoritmo de seleção clonal.

Por fim, Costa (2012) utilizou os trabalhos anteriores como base para construir uma formulação multiobjetivo para o problema. O autor propôs a combinação de duas funções objetivo: maximização da velocidade média e minimização da variância da velocidade dos veículos. Para otimizar esse conjunto de funções, foi utilizado o algoritmo NSGA-II, além de conceitos de memória e vizinhança adaptativa.

1.1.4 Programação Semafórica em Tempo Real

Apesar de obterem bons resultados, os trabalhos de Oliveira (2009) e de Costa (2012), gastavam horas, ou até mesmo dias, para retornar boas soluções, devido ao uso do simulador de tráfego para avaliar as programações semafóricas candidatas. Isso impedia o uso dos processos propostos para otimização da programação semafórica em tempo real. Contudo, vários trabalhos tem sido feito com essa abordagem e ela já tem sido testadas em grandes cidades (PMV, 2015). Além disso, o próprio Manual Brasileiro de Sinalização de Trânsito (DENATRAN, 2014) já indica as regras e modos de operação desse processo.

Dentre os trabalhos com essa abordagem, destaca-se Pereira and Ribeiro (2007), que demonstra os testes de otimização da programação semafórica em tempo real realizados pela BHTRANS na cidade de Belo Horizonte. A partir de seus resultados foi possível observar que dos 20 trechos que participaram do experimento, 13 (65%) tiveram melhorias em relação a programação semafórica fixa. Além disso, o controle em tempo real diminuiu em 16,34% a média

do tempo de percurso das rotas pesquisadas.

Assim, comprova-se a importância de novas arquiteturas que sejam capazes de realizar esse processo e obterem bons resultados.

1.2 Objetivos

Tendo em vista as limitações dos trabalhos anteriores, baseados em simuladores de tráfego, o objetivo deste trabalho é construir métodos para obter programações semafóricas otimizadas e robustas em tempo real, de forma a minimizar os tempos gastos pelos veículos para percorrerem os trechos sob análise.

Para isso, foram definidos os seguintes objetivos específicos:

- Propor duas novas arquiteturas para lidar com o problema de otimização da programação semafórica, onde:
  - O simulador de tráfego seja utilizado apenas no início do processo de otimização, para construir os cenários de fluxos de veículos nos movimentos do trecho.
  - A avaliação das programações semafóricas candidatas seja feita a partir do uso dos modelos matemáticos propostos por Akçelik (1991) e Webster (DENATRAN, 2014). Dessa forma, espera-se que o processo de otimização se torne consideravelmente mais rápido em relação aos métodos que utilizam os simuladores de tráfego.
  - As soluções candidatas sejam avaliadas para vários cenários de fluxos nos movimentos. Assim, busca-se minimizar os impactos causados pelas incertezas e variações dos fluxos e garantir que as programações semafóricas obtidas sejam robustas.
- Avaliar as arquiteturas com uso de três algoritmos de otimização mono-objetivo e um multiobjetivo;
- Testar as arquiteturas para cenários que simulam trechos reais.

1.3 Organização da Dissertação

Esta dissertação está organizada na seguinte ordem. A descrição do problema tratado neste trabalho é exibida no Cap. 2. Os conceitos sobre otimização evolucionária e seus algoritmos são abordados no Cap. 3. No Cap. 4, está detalhado o desenvolvimento das arquiteturas de otimização da programação semafórica em tempo real propostas neste trabalho. Os resultados dos experimentos realizados são exibidos no Cap. 5. Por fim, as conclusões sobre o trabalho são apresentadas no Cap. 6.

Capítulo 2

Programação Semafórica

Cruzamentos ou interseções podem conter movimentos conflitantes entre si. Isso ocorre quando os movimentos se cruzam e não podem ser realizados simultaneamente. É fundamental gerenciar esses conflitos para evitar acidentes. Em vias de menor volume de tráfego, é suficiente utilizar placas "DÊ A PREFERÊNCIA" ou "PARE' nas vias transversais, indicando a contenção do fluxo naquele ponto. Entretanto, em vias com maior fluxo de veículos, a espera dos veículos pode gerar atrasos, sendo necessário o uso de semáforos.

Os semáforos são dispositivos de controle do tráfego que alternam o direito de passagem entre veículos e/ou pedestres a partir de indicações luminosas. Seu objetivo principal é autorizar ou proibir a passagem dos veículos que desejam realizar um movimento. Para tanto, utiliza-se as cores verde e vermelho, respectivamente. Para evitar interrupções bruscas, utiliza-se também um tempo de atenção (amarelo), indicando ao motorista sobre a proximidade da mudança para o estado vermelho.

Segundo DENATRAN (2014), a programação semafórica consiste em determinar:

- a sequência e a dimensão dos estágios, correspondentes aos tempos de verde, amarelo e vermelho dos semáforos que controlam os movimentos;
- o tempo máximo e total dos ciclos, calculado como o somatório dos tempos dos estágios dos semáforos;
- a sincronização entre os ciclos (defasagens), quando for necessário, para evitar que os veículos parem em semáforos sequenciais existentes nos corredores de tráfego.

Em geral, a programação semafórica deve ser feita de forma adequada, com o intuito de minimizar o tempo médio (atraso) que os veículos gastam para percorrer os trechos em análise. Configurações inadequadas podem implicar em problemas como a geração de filas, atrasos, aumento da poluição, redução da segurança, dentre outros (Costa, 2012).

Algumas das principais referências utilizadas na engenharia de tráfego são: os Manuais de Semáforos (DENATRAN, 1984, 2014), o *Highway Capacity Manual* 2010 (HCM-2010) (TRB, 2010) e o Manual Brasileiro de Sinalização de Trânsito (CONTRAN, 2012). Uma descrição extensiva do problema abordado pode ser encontrada nesses documentos.

2.1 Movimentos

O termo **movimento** é usado para identificar a origem e o destino de veículos ou pedestres (DENATRAN, 2014). Algumas de suas definições merecem destaque:

Movimentos convergentes: movimentos com origens diferentes e mesmo destino;
Movimentos divergentes: movimentos com uma mesma origem e destinos diferentes;
Movimentos de entrada: movimentos que devem ser realizados para entrar em um trecho;
Grupo de movimentos: conjunto de múltiplos movimentos realizados simultaneamente.

A Figura 2.1 apresenta um cruzamento entre duas ruas de sentido único, onde existem quatro movimentos possíveis.

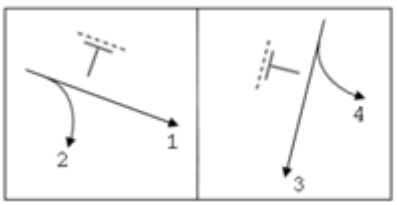


Figura 2.1: Exemplo de movimentos em um cruzamento.

2.2 Fluxo de Veículos

O **fluxo** de veículos, ou volume de tráfego, é o número de veículos que passam por uma determinada via, ou movimento, em um determinado período de tempo (CONTRAN, 2012). O **fluxo de saturação** corresponde ao número máximo de veículos que conseguiriam passar por um movimento no mesmo período de tempo. Por sua vez, o **fluxo de entrada** de veículos corresponde ao número veículos que adentram o trecho, por um movimento de entrada, durante o período de análise (DENATRAN, 2014).

Na prática, os fluxos de veículos nos movimentos são dados a partir da contagem, visual ou por aparelhos, efetuada pelas empresas de tráfego em determinados períodos de tempo. No entanto, nem sempre os dados reais estão disponíveis e, com isso, são utilizados simuladores de trafego para geração desses fluxos. Assim, é possível avaliar ferramentas de programação, estimar cenários em redes a serem instaladas/expandidas ou gerar cenários alternativos de fluxos de veículos em determinados trechos para auxiliar no processo de otimização.

Como exemplo, a aplicação de um simulador no cruzamento da Figura 2.1 deve retornar quantos veículos realizam cada um dos quatro movimentos permitidos naquele trecho. A quantidade de veículos deve ser condizente com o total de veículos que entra no trecho e atender todas as restrições de balanceamento (todos os veículos que chegam ao cruzamento pela avenida horizontal devem seguir 1 ou 2 e todos os veículos que chegam pela avenida vertical devem seguir 3 ou 4).

2.3 Grau de Saturação em um Movimento

Vilanova (2005) define o grau de saturação (a) em um movimento como a relação entre o número de veículos (fluxo) que chegam a um movimento e o número máximo de veículos que conseguem de fato passar por ele (fluxo de saturação) em um período de tempo. Assim, dizer que o grau de saturação do movimento é de 100% ($a = 1$) é o mesmo que dizer que o tempo de verde do semáforo que o controla é exatamente o suficiente para escoar todos os veículos que chegam a ele. Quando o grau de saturação do movimento se torna superior a 1 ($a > 1$), existe a formação de filas, que podem acarretar em engarrafamentos.

Também é importante ressaltar que nem todo o tempo de verde é utilizado de forma efetiva. Como pode ser visto na Figura 2.2, os veículos levam algum tempo para começar seu deslocamento e acelerar até sua velocidade normal de viagem no início do período de verde, o que indica um atraso inicial. Depois de alguns segundos, a fila de veículos descarrega a uma taxa constante, até que o semáforo passe para o estado amarelo, onde os veículos iniciam a desaceleração.

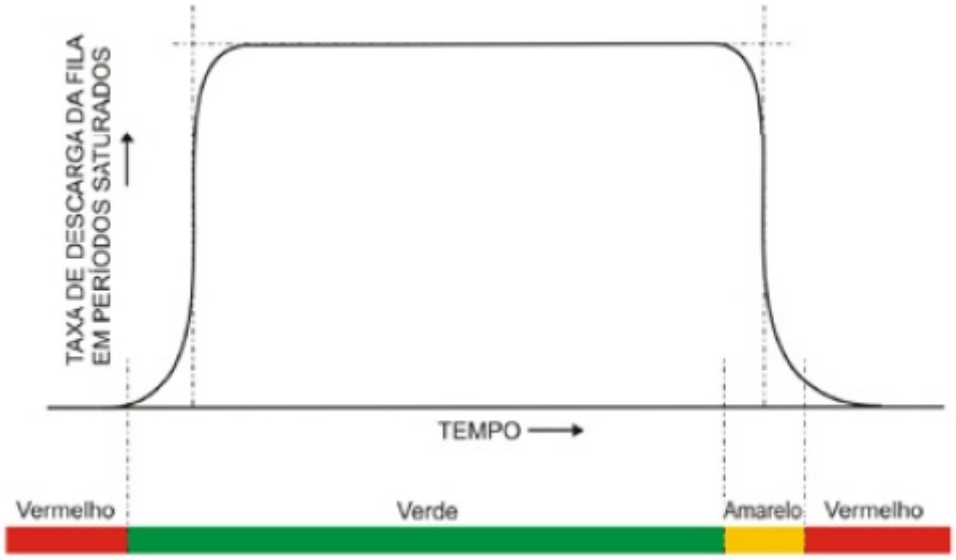


Figura 2.2: Comportamento do fluxo de veículos em um movimento, extraído de (Oliveira, 2009)

Com isso, Vilanova (2005) propõe um modelo matemático, exibido na Equação (2.1), que possibilita o cálculo do tempo de ciclo e dos tempos de verde através do método do grau de saturação. Esse modelo escolhe inicialmente qual o grau de saturação (a) desejado para os movimentos e, em seguida, calcula o tempo de ciclo (t_{ciclo}) e os tempos de verdes (t_{verde}) que o acarretarão, dado que os fluxos de veículos nos movimentos (q) e os fluxos de saturação (Q) são conhecidos.

$$a = \frac{q \times t_{ciclo}}{Q \times t_{verde}} \tag{2.1}$$

onde:

a é o grau de saturação do movimento;

q é o fluxo de veículos no movimento (V/h);

Q é o fluxo de saturação do movimento (V/h);

t_{verde} é o tempo de verde dos semáforos (s);
t_{ciclo} é o tempo de ciclo ao qual o semáforo que rege o movimento faz parte (s).

De acordo com Vilanova (2005), deve-se trabalhar com um grau de saturação entre 80% a 90% ($0,80 \leq a \leq 0,90$). Nessa faixa obtêm-se os menores valores para o atraso total do semáforo, visto que o tempo de verde seria suficiente para permitir o escoamento de todo o fluxo de veículos, mesmo que ocorra algum pequeno incidente. Porém, nem sempre é possível utilizar essa faixa de valores para todos os movimentos do trecho sob estudo. Logo, os projetistas devem considerar fatores locais que podem vir a influenciar na decisão de trabalhar com vias mais ou menos saturada, tais como:

- tempo máximo de ciclo;
- importância relativa daquela rua no sistema viário;
- volume de ônibus e caminhões;
- necessidade de deixar uma folga maior para vias onde incidentes são mais frequentes ou causam maiores implicações;
- risco que a fila formada pelo semáforo venha a bloquear outras vias.

2.4 Atraso em um Movimento

Quando é realizada uma programação semafórica, espera-se que ela impacte o mínimo possível no tempo de viagem dos veículos. Porém, como visto anteriormente, os projetistas considerar que os atrasos nos tempos de viagem são inevitáveis. De acordo com DENATRAN (2014), o **atraso** representa a diferença entre o tempo gasto por um veículo para percorrer um determinado trecho sob o controle semafórico e o tempo que ele gastaria se percorresse o mesmo trecho, sem interrupções, na velocidade máxima permitida.

Com isso, surgiram modelos matemáticos determinísticos e simuladores de tráfego capazes de calcular o atraso dos veículos nos movimentos e trechos. Isso permitiu o estudo dos impactos das mudanças nas programações semafóricas e, até mesmo, a otimização das mesmas, conforme discutido na Seção 1.1. No entanto, o uso de simuladores de tráfego para medir os atrasos dos veículos é um processo lento, o que impossibilita algumas aplicações práticas, como o processo de otimização da programação semafórica em tempo real.

Dentre os modelos matemáticos existentes na literatura, este trabalho utilizou os modelos propostos por Akçelik (1991) e Webster (DENATRAN, 2014), apresentados na sequência.

2.4.1 Modelo de Akçelik

Conforme descrito na Seção 1.1.1, Davidson (1966, 1978) propôs um modelo matemático determinístico para cálculo do tempo médio de viagem dos veículos. Porém, o mesmo foi alvo de discussões devido a problemas para calibrar seus parâmetros, visto que eles não tinham conceitos bem definidos (Rose et al., 1989).

Akçelik (1991) propôs um modelo alternativo que calcula o atraso médio dos veículos que passam por um movimento (Equação (2.2)) com o intuito de resolver esses problemas. O mesmo foi derivado do modelo de Davidson (1966) a partir do uso da técnica de transformação de coordenadas, cuja os detalhes podem ser vistos em Akçelik (1981).

$$t_m^{akcelik} = t_m^0 + 0.25 \times T_f \times \left[z + \left(z^2 + \frac{8 \times J_m \times a}{Q \times T_f} \right)^{0.5} \right] \tag{2.2}$$

onde:

t_m é o atraso médio dos veículos no movimento m (s);

t_m^0 é o tempo mínimo de viagem no movimento (s);

J_m é o parâmetro de qualidade do movimento;

a é o grau de saturação do movimento, calculado pela Equação (2.1);

$z = (x - 1)$;

T_f é o tempo total de simulação (s).

No entanto, o modelo de Akçelik (1991) depende do tempo mínimo de viagem do movimento (t_m^0), que é o tempo gasto por um veículo para percorrer o movimento na velocidade máxima permitida, sem qualquer impedimento. Além disso, ele utiliza o parâmetro de qualidade do movimento (J_m), que especifica fatores como interrupções na via e dependências entre os semáforos.

2.4.2 Modelo de Webster

Webster foi um dos pioneiros dos estudos sobre programação semafórica. Assim, ele também sugeriu um modelo, descrito em DENATRAN (2014), para calcular o atraso médio dos veículos que passam por um movimento (Equação (2.3)). Porém, seu modelo depende de condições não congestionadas e que a chegada dos veículos obedeça a uma distribuição aleatória em torno de um mesmo valor médio. Ou seja, a chegada não deve ser influenciada por sinalizações semafóricas a montante ou quaisquer outras condições que causem a alteração do valor médio de chegada ao longo do tempo.

$$t_m^{webster} = \frac{t_{ciclo}(1-p)^2}{2(1-p\times a)} + \frac{a^2}{2b(1-a)} - 0.65\left(\frac{t_{ciclo}}{b^2}\right)^{1/3} a^{(2+5p)} \tag{2.3}$$

onde:

t_m é o atraso médio de um veículo no movimento m (s);

t_{ciclo} é o tempo de ciclo ao qual o semáforo que rege o movimento faz parte (s);

p é a fração de verde (relação entre o tempo de verde e o tempo de ciclo);

a é o grau de saturação do movimento, calculado pela Equação (2.1);

$b = q/3600$ é o fluxo de veículos no movimento por segundo (V/s).

Nesse modelo, a primeira parcela corresponde ao atraso uniforme, a segunda a um atraso aleatório e a terceira consiste em um termo de ajuste da formulação teórica aos resultados registrados. Nas situações de trânsito livre, onde todos os veículos conseguem passar no primeiro período de verde, o atraso é composto pelas três parcelas. Quando a operação atinge o congestionamento total, o atraso aleatório desaparece, permanecendo os outros dois.

2.4.3 Atraso Total no Trecho

Considerando que o objetivo deste trabalho é construir métodos que possibilitem a otimização da programação semafórica de um trecho e não de apenas um ciclo isolado, torna-se necessário calcular o atraso total dos veículos no trecho em estudo. Assim, o atraso total t, a ser utilizado para avaliação das programações semafóricas candidatas, foi definido como o somatório dos atrasos dos veículos em todos os movimentos possíveis no trecho em estudo, conforme descrito na Equação (2.4).

$$t = \sum_{m \in \mathcal{M}} t_m \tag{2.4}$$

onde:

t é o atraso total dos veículos no trecho (s);

t_m é o atraso médio dos veículos em cada movimento (s), obtidos a partir dos modelos matemáticos determinísticos propostos por Akçelik (1991) e Webster (DENATRAN, 2014);

$\mathcal{M}$ é o conjunto de movimentos possíveis no trecho.

Por não considerar a defasagem entre ciclos, não é possível garantir a sincronização dos semáforos.

Capítulo 3

Algoritmos de Otimização

Neste capítulo são apresentados os algoritmos utilizados ao longo deste trabalho para solução do problema de otimização da programação semafórica. Foram testados cinco algoritmos metaheurísticos:

- Algoritmo Genético (AG) (Holland, 1975);
- *Particle Swarm Otmization* (PSO) (Kennedy and Eberhart, 1995);
- *Differential Evolution* (DE) (Storn and Price, 1997);
- *Nondominated Sorting Genetic Algorithm* II (NSGA II) (Deb et al., 2002);
- *Variable Neighborhood Search* (VNS) (Mladenović and Hansen, 1997).

O último algoritmo foi utilizado para refinamento das soluções obtidas pelos três primeiros.

3.1 Algoritmos Evolucionários Mono-objetivo

Segundo Goldberg (1989), os algoritmos evolucionários são algoritmos estocásticos inspirados na teoria da evolução de Charles Darwin que trabalham sobre uma população de soluções candidatas e simulam as características de evolução das espécies, como a seleção, o cruzamento e a mutação. Assim, de acordo com Fonseca (1995), a melhoria da população ocorre devido à repetida seleção dos indivíduos mais aptos e reprodução dos mesmos. As principais etapas dos AEs e os algoritmos utilizados neste trabalho são apresentados a seguir.

Algoritmo

O Algoritmo 1 apresenta a estrutura básica de um AE.

Algoritmo 1 Algoritmos Evolucionários

1: $pop^0 \leftarrow$ gerar população inicial($N_{individuos}$);
2: $fit^0 \leftarrow$ avaliar(pop^0);
3: $ger \leftarrow 1$;
4: **while** não antingiu condição de parada **do**
5: $\quad pop^{ger} \leftarrow$ criar próxima população(pop^{ger-1});
6: $\quad fit^{ger} \leftarrow$ avaliar(pop^{ger});
7: $\quad ger \leftarrow ger + 1$;
8: $\quad$ **return** melhor indivíduo $\in pop^{ger}$;
9: **end while**

onde:

$N_{individuos}$ é o número de indivíduos da população;

fit é o valor de *fitness* calculado a partir da avaliação dos indivíduos da população.

A primeira população nos AEs geralmente é composta por indivíduos gerados de forma aleatória. No entanto, deve-se seguir as regras de forma a respeitar a faixa de valores possíveis para cada variável e evitar a criação de indivíduos inválidos (von Zuben, 2009).

A avaliação dos indivíduos da população é feita por meio da função objetivo. Já a seleção é responsável por escolher os indivíduos mais aptos (pais) que farão parte dos processos seguintes que criarão as próximas gerações. Os pais são submetidos a mecanismos capazes de modificar a população para as próximas gerações. Os operadores, que modificam os indivíduos da população geralmente são divididos em operadores de cruzamento ou de mutação. No cruzamento, dois ou mais indivíduos pais são combinados para criação de novos filhos. Já a mutação é responsável por modificar os indivíduos de acordo com alguma regra probabilística, de forma a inserir novas informações na população.

O critério de parada varia de problema para problema. Geralmente os critérios mais adotados estão relacionados ao orçamento computacional do algoritmo, seja limitando o número de iterações do mesmo ou seu tempo de execução.

3.1.1 Algoritmos Genéticos

O Algoritmo Genético (AG) foi proposto por Holland (1975) e é considerado um dos marcos iniciais do desenvolvimento de AEs. O Algoritmo 2 apresenta as principais etapas do AG. As operações do AG utilizadas ao longo deste trabalho são descritas na sequência.

Algoritmo 2 Algoritmo Genético

1: $pop^0 \leftarrow$ geração da população inicial($N_{individuos}$);
2: $fit^0 \leftarrow$ avaliação(pop^0);
3: $ger \leftarrow 1$;
4: **while** não antingiu condição de parada **do**
5: $\quad U \leftarrow$ seleção(pop^{ger-1}, fit^{ger-1});
6: $\quad pop^{ger} \leftarrow$ cruzamento(U);
7: $\quad pop^{ger} \leftarrow$ mutação(pop^{ger});
8: $\quad fit^{ger} \leftarrow$ avaliação(pop^{ger});
9: $\quad ger \leftarrow ger + 1$;
10: $\quad$ **return** melhor indivíduo $\in pop^{ger}$;
11: **end while**

onde:

U é a população intermediária gerada após o processo de seleção dos indivíduos;

$N_{individuos}$ é o número de indivíduos da população.

Seleção

Após a avaliação dos indivíduos é feita a seleção dos pais que farão parte dos processos de cruzamento e mutação. Um dos métodos de seleção mais conhecidos é a Roleta Estocástica (do inglês, *Roulette Wheel Selection* ou RWS) (Goldberg, 1989). Esse método consiste em uma sequência de $N_{individuos}$ eventos de seleção independentes, onde a probabilidade de cada indivíduo ser selecionado é proporcional a sua *fitness* relativa.

No entanto, a roleta estocástica permite que os indivíduos com maior *fitness* se tornem muito dominantes no processo de seleção, o que diminui a diversidade das próximas gerações. Para lidar com esse problema, Baker (1987) propôs um método de seleção chamado Amostragem Universal Estocástica (do inglês, *Stochastic Universal Sampling* ou SUS). Nele, a roleta é preenchida da mesma forma que na Roleta Estocástica, com a área do *slot* referente a cada indivíduo sendo proporcional à sua *fitness*. Porém, são utilizados $N_{individuos}$ ponteiros igualmente

espaçados, fazendo com que toda a população seja escolhida em um único evento de seleção (veja exemplo da Figura 3.1).

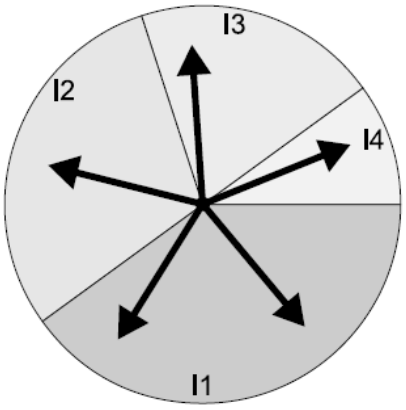


Figura 3.1: Método de seleção SUS, extraído de (Carrano, 2007).

Detalhes sobre outros métodos de seleção podem ser encontrados em Goldberg (1989); Fonseca (1995); Tanomaru (1995); Soares (1997).

Cruzamento

Uma vez selecionados os indivíduos pais, o AG passa à etapa de cruzamento. Neste trabalho foram utilizados dois tipos de cruzamento: o cruzamento de dois pontos e o cruzamento uniforme (Goldberg, 1989). O operador de cruzamento de dois pontos (*two points crossover*) consiste em escolher dois pontos aleatórios dentro do comprimento do indivíduo. Então, as posições entre os dois pontos são transferidas de um pai para outro. A Figura 3.2 exibe um exemplo desse cruzamento.

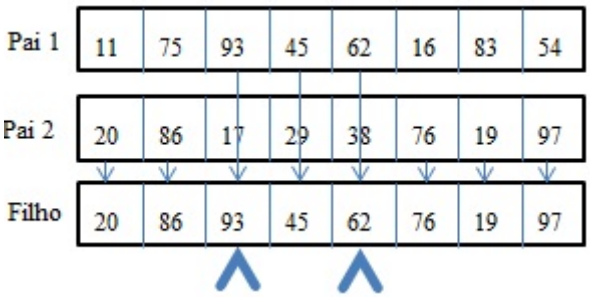


Figura 3.2: Exemplo de cruzamento de dois pontos.

O cruzamento uniforme (*uniform crossover*) sorteia com probabilidade uniforme quais posições do indivíduo filho serão herdadas de cada pai, conforme mostrado na Figura 3.3.

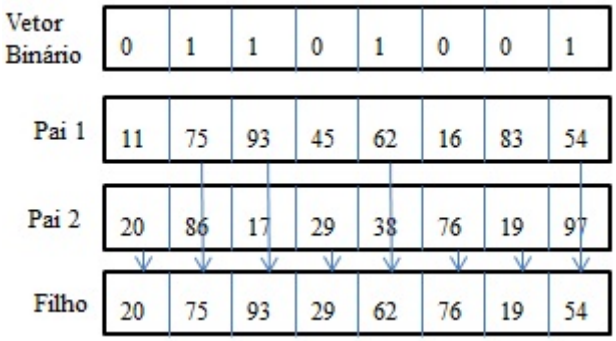


Figura 3.3: Exemplo de cruzamento uniforme.

Outros operadores de cruzamento para o AG podem ser encontrados em Ackley (1987); Goldberg (1989); Eshelman et al. (1989); Davis and Principe (1991); Syswerda (1989); Radcliffe (1990); Wright (1991)

Mutação

Após o cruzamento, é realizada a etapa de mutação, responsável por modificar os indivíduos e, consequentemente, aumentar a diversidade da população. O operador de mutação empregado neste trabalho é baseado no operador clássico conhecido como *Flip*. Este sorte aleatoriamente posições do indivíduo que serão alteradas. Essa alteração se dá pela substituição do valor presente por algum outro valor válido para o domínio da variável. Um exemplo deste operador é apresentado na Fig. 3.4.

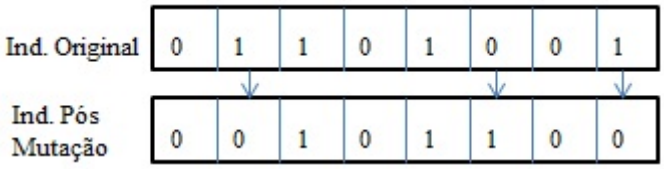


Figura 3.4: Exemplo de mutação *Flip* em indivíduo com codificação binária.

Outros operadores de mutação em AGs podem ser encontrados em Goldberg (1989); Tanomaru (1995); Soares (1997); Ramos et al. (2003).

3.1.2 *Particle Swarm Otmization*

O *Particle Swarm Optimization* (PSO) (Kennedy and Eberhart, 1995) é um algoritmo evolucionário baseado no comportamento social coletivo, como o de revoadas de pássaros ou o de cardumes de peixes. Ele compartilha algumas características com os AGs, como a inicialização da população e os critérios de parada. Porém, o PSO não possui seleção nem operadores de cruzamento explícitos, o que simplifica sua implementação. O processo de mutação, por sua vez, é realizado com uso da partícula de velocidade (v). O algoritmo detalhado é exibido no Algoritmo 3 e seu processo de mutação é apresentado na sequência.

Algoritmo 3 *Particle Swarm Optimization*

1: $pop^0 \leftarrow$ geração da população inicial(N);
2: $fit^0 \leftarrow$ avaliação(pop^0);
3: $v^0 \leftarrow 0$;
4: $ger \leftarrow 1$;
5: **while** não antingiu condição de parada **do**
6: $\quad (pop^{ger}, v^{ger}) \leftarrow$ mutação(pop^{ger-1}, v^{ger-1});
7: $\quad fit^{ger} \leftarrow$ avaliação(pop^{ger});
8: $\quad ger \leftarrow ger + 1$;
9: **end while**
10: **return** melhor indivíduo $\in pop^{ger}$

Mutação

O processo de mutação do PSO (Algoritmo 4) é feito a partir da partícula de velocidade, que define a direção em que os indivíduos devem seguir para convergência. A partícula de velocidade (v_i), exibida na Equação (3.1), é influenciada pelas melhores soluções já obtidas em cada posição ($pop_{i_{best}}$), além da melhor solução global encontrada até o momento (pop_{best}). Dessa forma, espera-se que os indivíduos sejam iterativamente direcionados para posições mais promissoras no espaço de busca, conforme a Equação (3.2).

$$v_i \leftarrow \omega v_i + \phi_l(pop_{i_{best}} - pop_i) + \phi_g(pop_{best} - pop_i) \tag{3.1}$$

$$pop_i = pop_i + v_i \tag{3.2}$$

onde:

pop_i é o indivíduo atual na posição i a sofrer mutação;

v_i é a partícula de velocidade utilizada para atualizar o indivíduo;

pop_{best} é o melhor indivíduo já encontrado no processo de otimização;

$pop_{i_{best}}$ é o melhor indivíduo já encontrado no processo de otimização na posição i;

ω, ϕ_l e ϕ_g são parâmetros selecionados de forma a controlar o comportamento e a eficácia do algoritmo.

Algoritmo 4 PSO - Mutação

```
1: function MUTAÇÃO(pop, v)
2:     for i = 1 até N do
3:         v_i ← ωv_i + φ_l(pop_{i_best} − pop_i) + φ_g(pop_best − pop_i);
4:         for j = 1 até N_posicoes do
5:             if random ≤ p_mut then
6:                 U_{i_j} = pop_{i_j} + v_{i_j}
7:             end if
8:         end for
9:     end for
10:    return (U, v);
11: end function
```

onde:

$N_{posicoes}$ é o número de posições no indivíduo;

U é a população gerada pelo processo de mutação;

p_{mut} é a probabilidade da posição do indivíduo sofrer mutação, ou seja, ser influenciada pela partícula de velocidade.

Por utilizar a partícula de velocidade, o PSO torna-se eficiente para problemas monomodais. No entanto, isso o deixa suscetível à convergência prematura para ótimos locais nos problemas multimodais (van den Bergh and Engelbrecht, 2010; Bonyadi and Michalewicz, 2014). Assim, diversos estudos têm sido feitos em relação ao ajuste dos parâmetros de comportamento da partícula de velocidade para evitar esse problema (van den Bergh, 2001; Clerc and Kennedy, 2002; Trelea, 2003). Além do ajuste dos parâmetros, outras linhas de pesquisa também sugerem:

- Causar pertubações na partícula, tais como multiplicar seus termos por fatores aleatórios (Evers, 2009; Lovbjerg and Krink, 2002; Xinchao, 2010; Xie et al., 2005);

- Adaptar os parâmetros de comportamento dinamicamente (Zhan et al., 2009);
- Utilizar mais de uma população, de forma a aumentar a abrangência de busca do método (Cheung et al., 2014). Isso também possibilita o uso do PSO para problemas de otimização multiobjetivo (Nobile et al., 2012).

3.1.3 *Differential Evolution*

O último algoritmo evolucionário mono-objetivo utilizado foi *Differential Evolution* (DE), proposto por Storn and Price (1997). Sua estrutura também é semelhante à do AG, conforme pode ser visto no Algoritmo 5. A principal diferença entre os algoritmos está no processo de geração das próximas populações. Inicialmente é gerada uma população intermediária, a partir do processo de mutação diferencial. Então, é realizado o cruzamento entre a população atual e a população intermediária, utilizando os indivíduos nas mesmas posições. Outros detalhes sobre seu processo de mutação diferencial são descritos a seguir.

Algoritmo 5 *Differential Evolution*

1: $pop^0 \leftarrow$ geração da população inicial(N);
2: $fit^0 \leftarrow$ avaliação(pop^0);
3: $ger \leftarrow 1$;
4: **while** não antingiu condição de parada **do**
5: $\quad U \leftarrow$ mutação diferencial(pop^{ger-1});
6: $\quad pop^{ger} \leftarrow$ cruzamento(pop^{ger-1}, U);
7: $\quad fit^{ger} \leftarrow$ avaliação(pop^{ger});
8: $\quad ger \leftarrow ger + 1$;
9: **end while**
10: **return** melhor indivíduo $\in pop^{ger}$

onde:

U é a população intermediária gerada pelo processo de mutação.

Mutação Diferencial

A principal característica do DE é seu processo de mutação diferencial, que é realizado para todos os indivíduos da população atual, sem seleção prévia. O operador de mutação diferencial mais utilizado é o *random-to-random*, definido na Equação (3.3). Além dele, também foram utilizados neste trabalho os operadores *current-to-random* (Equação (3.4)) e o operador *current-*

to-best (Equação (3.5)) (Das and Suganthan, 2011), com o intuito de avaliar aquele que seria o melhor para o problema tratado.

$$pop_i = pop_{r1} + \mu \times (pop_{r2} - pop_{r3}) \tag{3.3}$$

$$pop_i = pop_i + \mu \times (pop_{r1} - pop_{r2}) \tag{3.4}$$

$$pop_i = pop_i + \mu \times (pop_{best} - pop_r) \tag{3.5}$$

onde:

pop_i é o indivíduo atual a sofrer cruzamento;

pop_{best} é o melhor indivíduo já encontrado no processo de otimização;

pop_r são indivíduos escolhidos de forma aleatória;

μ é o peso diferencial e deve estar no intervalo $\mu \in [0, 2]$.

O processo de mutação completo pode ser visto no Algoritmo 6.

Algoritmo 6 DE - Mutação Diferencial

```
1: function MUTAÇÃO DIFERENCIAL(pop)
2:     for i = 1 até N_individuos do
3:         (pop_r1, pop_r2, pop_r3) ← choose(pop);
4:         for j = 1 até N_posicoes do
5:             if random ≤ p_mut then
6:                 U_ij ← pop_r1_j + μ × (pop_r2_j − pop_r3_j);
7:             end if
8:         end for
9:     end for
10:    return U;
11: end function
```

onde:

$N_{posicoes}$ é o número de posições no indivíduo;

U é a população intermediária gerada pelo processo de mutação;

pop_r são os indivíduos selecionados aleatoriamente na população;

p_{mut} é a probabilidade da posição do indivíduo sofrer mutação diferencial.

É possível encontrar na literatura outros exemplos de operadores de mutação diferencial, como os operadores baseados em centroides (Ali et al., 2011) e os baseados em lógica *fuzzy* (dos Santos et al., 2015). Para outros exemplos, recomenda-se a leitura de Price et al. (2006), Das and Suganthan (2011) e Mallipeddia et al. (2011).

3.2 Algoritmos Evolucionários Multiobjetivo

A otimização multiobjetivo é um tópico de pesquisa muito importante para cientistas e engenheiros, uma vez que a maioria dos problemas dependem do atendimento de dois ou mais critérios (Coello, 1999b).

Deb (2008) define os problemas de otimização multiobjetivo como sendo aqueles que envolvem mais de uma função objetivo, que devem ser minimizadas ou maximizadas simultaneamente. Assim, a principal diferença desses problemas para os problemas mono-objetivos é que eles geralmente não possuem apenas uma solução ótima, mas um conjunto de soluções eficientes, chamado conjunto Pareto-ótimo, no qual as soluções competem de acordo com o ganho ou perda (*trade-off*) em determinados objetivos.

Pode-se citar alguns Algoritmos Evolucionários Multiobjetivo:

- Sem elitismo:
  - *Multiobjective Genetic Algorithm* (MOGA) (Fonseca and Fleming, 1993);
  - *Niched-Pareto Genetic Algorithm* (NPGA) (Horn et al., 1994);
  - *Nondominated Sorting Genetic Algorithm* (NSGA) (Srinivas and Deb, 1994).
- Com elitismo:
  - *Pareto Archived Evolution Strategy* (PAES) (Knowles and Corne, 1999);
  - *Strength Pareto Evolutionary Algorithm* (SPEA) (Zitzler and Thiele, 1999);
  - *Pareto Envelope-based Selection Algorithm* (PESA) (Corne et al., 2000);
  - *Strength Pareto Evolutionary Algorithm* II (SPEA-II) (Zitzler et al., 2001);
  - *Nondominated Sorting Genetic Algorithm* II (NSGA-II) (Deb et al., 2002);
  - *Multiobjective Evolutionary Algorithm Based on Decomposition* (MOEA/D) (Zhang and Li, 2007);

- *Multiobjective Evolutionary Algorithm Based on Decomposition and Differential Evolution* (MOEA/D-DE) (Li and Zhang, 2009);
- *Nondominated Sorting Genetic Algorithm* III (NSGA-III) (Yuan et al., 2014).

Alguns conceitos sobre os AEs multiobjetivo e o NSGA-II, utilizado neste trabalho, são exibidos a seguir.

Fronteiras Pareto

De acordo com Takahashi (2007), o objeto fundamental da otimização multiobjetivo é encontrar o conjunto Pareto-ótimo, que contém as soluções ótimas para o problema. Esse conceito foi formulado por Vilfredo Pareto no século XIX e constituiu o início da pesquisa em otimização multiobjetivo (Coello, 1999a). Takahashi (2007) define os conceitos de solução Pareto-ótima e de dominância, conforme as definições 1 e 2, respectivamente.

Definição 1 *(Solução Pareto-ótima).* $y^* \in Y$ *é uma solução Pareto-ótima (eficiente, não-dominada, não inferior) se não existir nenhuma outra solução factível* y *no espaço de soluções* Y *que a domine.*

Definição 2 *(Dominância).* y_1 *domina* y_2 *se:*
$f_i(y_1) \leq f_i(y_2) \; \forall \; i \in (1, \ldots, |\mathcal{O}|)$ *e*
$\exists \; j \mid f_j(y_1) < f_j(y_2)$.

onde:
y_1 e y_2 são duas soluções no espaço de soluções Y;
$\mathcal{O}$ é o conjunto de objetivos tratados.

Logo, a solução y_1 domina a solução y_2 caso seja melhor em ao menos um objetivo e não seja pior em nenhum outro. A imagem do conjunto das soluções Pareto-ótimas no espaço dos objetivos é denominada Fronteira Pareto-ótima (ilustrada na Figura 3.5).

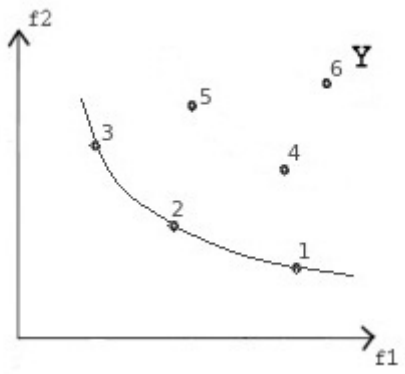


Figura 3.5: Exemplo de uma Fronteira Pareto-ótima.

Avaliação de Desempenho

A comparação de duas Fronteiras Pareto-ótimas obtidas pelos algoritmos de otimização multiobjetivo não é algo trivial. Dessa forma, foram criadas diversas métricas que devem ser utilizadas para avaliar o desempenho dos algoritmos de otimização multiobjetivo.

A métrica do hipervolume (ou *S–Metric*) foi proposta por Zitzler and Thiele (1998) e avalia a distribuição das soluções em relação ao espaço de busca. Assim, o hipervolume H de uma Fronteira Pareto-ótima determina a área coberta ou dominada pelo conjunto de soluções na mesma. Para calculá-lo é necessário definir um ponto de referência. Em problemas de maximização é comum utilizar o ponto $(0, 0)$. Já nos problemas de minimização é escolhido um ponto de referência (x, y), que determina os limites superiores de cada função objetivo. Com isso, quanto maior o hipervolume de uma Fronteira Pareto-ótima, melhor será sua qualidade. Um exemplo de hipervolume é a área sombreada exibida na Figura 3.6.

Outros exemplos de métricas para avaliação de desempenho dos algoritmos de otimização multiobjetivo são:

- cardinalidade (Hansen, 1998);
- distância média (Czyzźak and Jaszkiewicz, 1998);
- distância máxima (Knowles and Corne, 2002);
- *epsilon* (Fonseca et al., 2005).

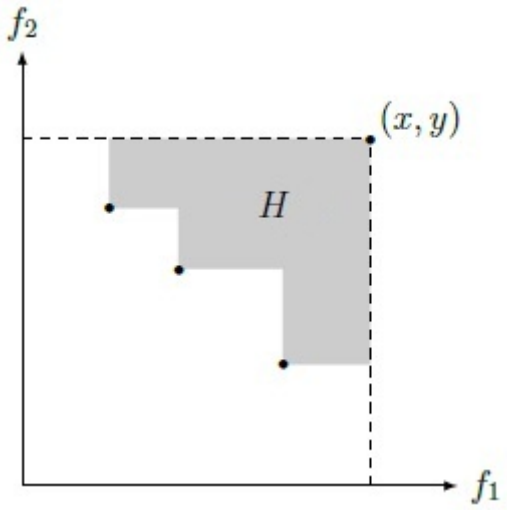


Figura 3.6: Hipervolume de uma Fronteira Pareto-ótima.

3.2.1 *Nondominated Sorting Genetic Algorithm* II

O *Nondominated Sorting Genetic Algorithm II* (NSGA-II) (Deb et al., 2002) foi proposto como uma versão melhorada no NSGA (Srinivas and Deb, 1994). Este visava tratar as principais deficiências do algoritmo anterior, a saber:

Alta complexidade computacional: O procedimento utilizado pelo NSGA para classificação das fronteiras não dominadas tinha complexidade computacional $O(|\mathcal{O}| \times N^3_{individuos})$, sendo $\mathcal{O}$ é o conjunto de objetivos tratados e $N_{individuos}$ é o tamanho da população. Já o *Fast Nondominated Sorting*, adotado pelo NSGA-II, possui complexidade $O(|\mathcal{O}| \times N^2_{individuos})$.

Dependência de parâmetros de nicho: O NSGA utilizava um método de nicho baseado em um parâmetro σ_{share}. No entanto, o ajuste desse parâmetro era em geral muito sensível ao problema tratado. O NSGA-II utiliza o *Crowding Distance Assignment*, que não depende de nenhum parâmetro, para manutenção de diversidade.

Abordagem não elitista: O NSGA não empregava elitismo. O NSGA-II é um algoritmo naturalmente elitista, o que tende a acelerar a convergência para melhores soluções.

A estrutura do NSGA-II é apresentada no Algoritmo 7. As principais etapas desse algoritmo são discutidas na sequência.

Algoritmo 7 NSGA-II

1: $pop^1 \leftarrow$ geração da população inicial($N_{individuos}$);
2: $fit^1 \leftarrow$ avaliação(pop^1);
3: $\mathcal{F}_1 \leftarrow$ encontrar pareto(pop^1, fit^1);
4: $pop^0 \leftarrow \emptyset$;
5: $fit^0 \leftarrow \emptyset$;
6: **for** $ger \leftarrow 1$ até $N_{geracoes}$ **do**
7: $V^{ger} \leftarrow pop_{ger} \bigcup pop^{ger-1}$;
8: $fit^{ger} \leftarrow fit^{ger} \bigcup fit^{ger-1}$;
9: $U^{ger} \leftarrow$ seleção($V^{ger}, fit^{ger}, N_{individuos}$);
10: $pop^{ger+1} \leftarrow$ cruzamento(U^{ger});
11: $pop^{ger+1} \leftarrow$ mutação(pop^{ger+1});
12: $fit^{ger+1} \leftarrow$ avaliação(pop^{ger+1});
13: $\mathcal{F}_{ger+1} \leftarrow$ encontrar pareto($\mathcal{F}_{ger} \bigcup pop^{ger+1}, fit^{ger+1}$);
14: **end for**

onde:

$\mathcal{F}_{ger}$ é a fronteira Pareto não dominada na geração ger;

V é a população composta pela união da população anterior com a atual;

U é a população intermediária gerada composta pelos indivíduos a serem utilizados como pais após a etapa de seleção.

Classificação das Fronteiras

O *Fast Nondominated Sorting* (FNS) é o método responsável por classificar os níveis de dominância (*rank*) de cada solução da população.

Inicialmente o conjunto $\mathcal{F}^1$ de soluções não dominadas é extraído da população inicial. Então, um novo conjunto $\mathcal{F}^2$ de soluções não-dominadas é extraído da população restante. O processo se repete até que não existam mais soluções na população. Assim, cada subconjunto da população, $\mathcal{F}^2, \ldots, \mathcal{F}^k$, contém indivíduos que são dominados apenas pelos indivíduos dos subconjuntos precedentes. O processo possui complexidade total $O(|\mathcal{O}| \times N^2_{individuos})$.

Este método é apresentado no Algoritmo 8. A Figura 3.7 ilustra um exemplo para um problema de minimização de duas funções objetivo.

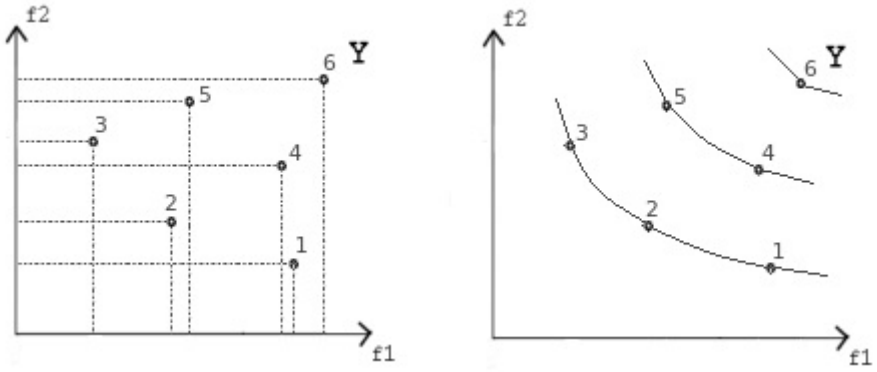


Figura 3.7: Exemplo de resultado do método *Fast Nondominated Sort*, extraído de (Carrano, 2007).

Algoritmo 8 NSGA-II - *Fast Nondominated Sort* (extraído de (Carrano, 2007))

for each $p \in pop$ **do**
  $\mathcal{Q}_p \leftarrow \emptyset$;
  $n_p \leftarrow 0$;
  for each $q \in pop$ **do**
    if $p < q$ **then**
      $\mathcal{Q}_p \leftarrow \mathcal{Q}_p \bigcup q$;
    else if $(q < p)$ **then**
      $n_p \leftarrow n_p + 1$;
    end if
  end for
  if $n_p = 0$ **then**
    $p_{rank} \leftarrow 1$;
    $\mathcal{F}^1 \leftarrow \mathcal{F}^1 \bigcup p$;
  end if
end for
$i \leftarrow 1$;
while $\mathcal{F}^i \neq \emptyset$ **do**
  $\mathcal{F}^{i+1} \leftarrow \emptyset$;
  for each $p \in \mathcal{F}^i$ **do**
    for each $q \in \mathcal{Q}_p$ **do**
      $n_q \leftarrow n_q - 1$;
      if $n_q = 0$ **then**
        $q_{rank} \leftarrow i + 1$;
        $\mathcal{F}^{i+1} \leftarrow \mathcal{F}^{i+1} \bigcup q$;
      end if
    end for
  end for
  $i \leftarrow i + 1$;
end while

onde:

$\mathcal{F}^i$ é a i-ésima fronteira;

n_p é o número de indivíduos que dominam o indivíduo p;

$\mathcal{Q}_p$ é o conjunto de indivíduos dominados pelo indivíduo p.

Preservação da Diversidade

O NSGA original utiliza a abordagem conhecida como função de compartilhamento para manter a diversidade em uma população. Esta depende da escolha do parâmetro σ_{share}, o que requer algum método de ajuste. Entretanto, no algoritmo NSGA-II, a função de compartilhamento foi substituída pelo procedimento *Crowding Distance Assignment* (CDA). Essa nova abordagem não requer nenhum parâmetro pré-definido para manter a diversidade dos indivíduos da população (Deb et al., 2002). Além disso, esse novo procedimento tem uma melhor complexidade computacional.

Seu primeiro passo é ordenar os indivíduos da população em ordem ascendente para cada uma das $|\mathcal{O}|$ funções objetivo tratadas. Depois disso, um valor teórico infinito é atribuído às soluções extremas (soluções com menores e maiores valores para cada uma das funções objetivo). Para todas as demais soluções são atribuídos um valor de distância igual à diferença absoluta normalizada nos valores de funções das duas soluções adjacentes. Este calculo é feito para todas as funções objetivo, conforme a Equação (3.6).

$$\mathcal{F}[i]_{distance} = \sum_{o \in \mathcal{O}} \left(\frac{\mathcal{F}[i+1]^o - \mathcal{F}[i-1]^o}{\mathcal{F}[max]^o - \mathcal{F}[min]^o} \right) \tag{3.6}$$

onde:

$\mathcal{F}[i]_{distance}$ *crowding distance* do indivíduo i na fronteira $\mathcal{F}$ tratada;

$\mathcal{O}$ conjunto das funções objetivo tratadas;

$\mathcal{F}^o_{max}$ e $\mathcal{F}^o_{min}$ são os valores máximo e minimo presentes na fronteira $\mathcal{F}$ para a o-ésima função objetivo.

O procedimento é exibido no Algoritmo 9 (extraído de (Carrano, 2007)). A Figura 3.8 mostra um exemplo para este cálculo, onde os pontos em negrito pertencem à primeira fronteira.

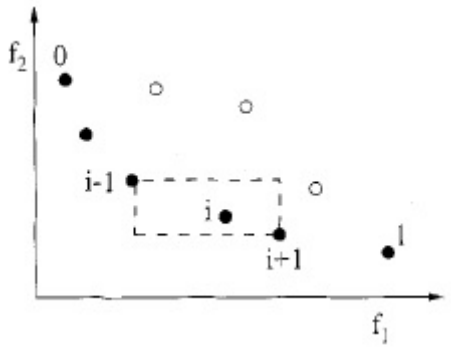


Figura 3.8: Exemplo do *Crowding Distance Assignment*.

Algoritmo 9 NSGA-II - *Crowding Distance Assignment*

1: $l \leftarrow |\mathcal{F}|$;
2: $\mathcal{F}[*]_{distance} \leftarrow 0$;
3: **for each** $o \in \mathcal{O}$ **do**
4: $\quad \mathcal{F} \leftarrow sort(\mathcal{F}, o)$;
5: $\quad \mathcal{F}[1]_{distance} \leftarrow \mathcal{F}[l]_{distance} \leftarrow \infty$;
6: $\quad$ **for** i **from** 2 **to** $l-1$ **do**
7: $\quad\quad \mathcal{F}[i]_{distance} \leftarrow \mathcal{F}[i]_{distance} + \frac{(\mathcal{F}[i+1]^o - \mathcal{F}[i-1]^o)}{\mathcal{F}^o_{max} - \mathcal{F}^o_{min}}$;
8: $\quad$ **end for**
9: **end for**

Com isso, a complexidade deste procedimento é definida pelo algoritmo de ordenação. Considerando o uso *Quick Sort* (Hoare, 1961), que é executado $|\mathcal{O}|$ vezes, já que as soluções devem ser ordenadas de forma independente para cada função objetivo, o CDA possui ordem de complexidade $O(|\mathcal{O}| \times N_{individuos} \times log(N_{individuos}))$.

Seleção

Assim como no AG, após a avaliação o NSGA-II realiza a seleção dos indivíduos que serão utilizados como pais das próximas populações. Para tal, ele utiliza o método *Stochastic Tournament* (Miller and Goldberg, 1995), utilizando o operador de precedência definido por Deb et al. (2002). Este operador é definido como:

Definição 3 *(Precedência).* $i \prec j$ *se:*
$(i_{rank} < j_{rank})$ *ou* $[(i_{rank} = j_{rank})$ *e* $(i_{distance} > j_{distance})]$.

onde:

i_{rank} é o *rank* de dominância do indivíduo i;

$i_{distance}$ é o *crowding distance* do indivíduo i.

Assim, entre duas soluções com diferentes *ranks*, aquela que possuir o menor *rank* é a escolhida. No entanto, se ambas soluções pertencerem à mesma fronteira, então a solução com maior *crowding distance* é a escolhida, para manutenção da diversidade.

Cruzamento e Mutação

Os operadores de cruzamento e mutação empregados no NSGA-II podem ser os mesmos utilizados em AGs mono-objetivos.

3.3 Metaheurísticas de Busca Local

As metaheurísticas de busca local são métodos que coordenam procedimentos de busca locais com estratégias que lhes permitem escapar de mínimos locais. Com isso, elas realizam uma busca robusta no espaço de soluções de um problema (Glover and Kochenberger, 2003).

Dentre as metaheurístisticas de busca local existentes, cita-se:

- *Best-first Search* (BFS) (Pearl, 1984);
- *Tabu Search* (Glover, 1986);
- *Simulated Annealing* (Kirkpatrick et al., 1983);
- *Variable Neighborhood Search* (VNS) (Mladenović and Hansen, 1997);
- *Greedy Randomized Adaptive Search Procedures* (GRASP) (Feo and Resende, 1989).

Alguns mecanismos de busca local e o VNS, utilizado neste trabalho, são descritos na sequencia.

Busca de Soluções Vizinhas

As metaheurísticas de busca local trabalham a partir de uma solução inicial x e realizam buscas em sua vizinhança a fim de se obter melhores soluções.

Um dos métodos utilizados para a busca é o *Best Improvement*, que consiste em testar todas as possíveis soluções dentro da vizinhança $V(x)$, a fim de encontrar a melhor vizinha x'. Esse processo é repetido iterativamente e termina quando nenhuma solução melhor é encontrada. Seu algoritmo é descrito no Algoritmo 10.

Algoritmo 10 *Best Improvement*

```
1: function BestImprovement(x, V)
2:     repeat
3:         x' ← x;
4:         x ← argmin(f(y)) ∀ y ∈ V(x);
5:     until f(x) ≥ f(x');
6:     return x;
7: end function
```

Uma alternativa, geralmente de menor custo, ao *Best Improvement* é o *First Improvement*. Ele retorna a primeira solução vizinha encontrada que seja melhor que a atual, sem ter necessidade de avaliar todas as soluções presentes na vizinhança. Seu algoritmo é descrito no Algoritmo 11.

Algoritmo 11 *First Improvement*

```
1: function FirstImprovement(x, V)
2:     repeat
3:         x' ← x;
4:         i ← 1;
5:         repeat
6:             x ← argmin(f(x), f(x^i)), x^i ∈ V(x);
7:             i ← i + 1;
8:         until f(x) < f(x^i) or i > |V(x)|;
9:     until f(x) ≥ f(x');
10:    return x;
11: end function
```

3.3.1 *Variable Neighborhood Search*

O *Variable Neighborhood Search* (VNS), proposto por Mladenović and Hansen (1997), é uma metaheurística voltada à solução de problemas de otimização combinatória e problemas de otimização global. Ela trabalha com mudanças das vizinhanças, com o objetivo de reduzir a probabilidade do algoritmo ficar preso em mínimos locais (um mínimo local sob uma função

de vizinhança pode não ser um mínimo local quando analisado por outra). A ideia central do VNS pode ser apresentada pelos seguintes passos:

1. A partir de uma solução, busca-se alguma outra que seja melhor dentro de sua vizinhança.
2. Se nenhuma solução melhor for encontrada, deve-se alterar a vizinhança, de forma a alterar o espaço de busca e fugir de possíveis ótimos locais.
3. Se ainda assim nenhuma solução melhor for encontrada, deve-se fazer alguma alteração (*shaking*) na solução atual e recomeçar o processo.

De acordo com Mladenović and Hansen (1997), o VNS é mais simples que outras metaheurísticas e não requer ajuste de parâmetros. Ainda assim, ela é capaz de prover boas soluções e de forma eficiente.

A estrutura do VNS é apresentada no Algoritmo 12. Os aspectos mais importantes são discutidos na sequência. Na literatura existem também diversas outras obras sobre o VNS. Dentre elas, cita-se (Gendreau and Potvin, 2010; Glover and Kochenberger, 2003; Burke and Kendall, 2014).

Algoritmo 12 VNS

```
1: function VNS(x, k_max, V)
2:     x' ← x;
3:     for k ← 1 até k_max do
4:         v ← 1;
5:         repeat
6:             x'' ← Search(x', V^v);
7:             NeighborhoodChange(x, x'', v, k);
8:         until v ≤ |V|;
9:         x' ← shaking(x, k);
10:    end for
11: end function
```

onde:

x é a solução atual;

k_max é o número máximo de modificações que podem ser feitas na solução atual para recomeçar o processo;

$\mathcal{V}$ é o conjunto de vizinhanças a serem utilizadas.

Mudança de Vizinhança:

Como dito anteriormente, as metaheurísticas devem prover mecanismos que possibilitem ao processo de busca escapar de ótimos locais. Assim, o VNS é implementado com k_max vizinhanças ordenadas, geralmente de forma crescente em relação a cardinalidade de seus conjuntos de soluções candidatas. Uma vez que o método de busca encontra uma solução melhor que a atual, ele permanece utilizando a mesma vinhança, pois existe a tendência de haver outras soluções melhores nas proximidades. Porém, quando ele não consegue evoluir, o método *Neighborhood Change*, descrito no Algoritmo 13, retorna a próxima vizinhança a ser utilizada, a fim de aumentar o espaço de busca e permitir que o VNS escape de um possível ótimo local.

Algoritmo 13 VNS - *Neighborhood Change*

```
1: function NeighborhoodChange(x, x', v, k)
2:     if f(x') < f(x) then
3:         x ← x';
4:         v ← 1;
5:         k ← 1
6:     else
7:         v ← v + 1;
8:     end if
9: end function
```

3.4 Tratamento de Restrições

Geralmente, as restrições são tratadas nas metaheurísticas de quatro formas:

Eliminação de Soluções

No método de eliminações de soluções, todas as soluções não factíveis são substituídas por outras, que estejam dentro da região factível do problema. Caso necessário, parte dessas novas soluções pode ser descartada por baixo desempenho. Porém, esta técnica só é viável para problemas com baixa probabilidade de geração de soluções infactíveis, uma vez que a mesma pode causar perda de diversidade da população.

Reparo de Soluções

O método de reparo de soluções propõe o ajuste das mesmas por meio de um algoritmo específico, pelo qual as soluções infactíveis do problema são deslocadas para a região factível. Em alguns casos, o reparo de soluções pode acarretar alto custo computacional, o que pode inviabilizar a aplicação dessa técnica.

Penalidade de Soluções

O método de penalidade de soluções consiste em relaxar as restrições do problema em termos que são acrescidos (ou multiplicados) na função objetivo. Geralmente os coeficientes dessas restrições são ajustados de forma a pequenas violações gerarem um aumento significativo do valor da função objetivo. Assim, Deb (2000) propôs o calculo da *fitness* do indivíduo x segundo a Equação (3.7).

$$fitness(x) = \begin{cases} f(x), \text{ caso } x \text{ seja factível} \\ f(x) + \sum_{g \in \mathcal{G}} \max(g(x), 0), \text{ caso contrário} \end{cases} \tag{3.7}$$

onde:

$f(x)$ é a função objetivo do problema para o indivíduo x;

$\mathcal{G}$ é o conjunto das funções penalidades aplicadas às restrição do problema.

Seleção por Torneio

Por fim, o método da seleção por torneio é muito utilizado em algoritmos de busca local. Ao comparar duas soluções, ele realiza um torneio com as seguintes regras (Deb, 2000):

1. Quando as duas soluções são factíveis, a que possuir o melhor valor de função objetivo $f(x)$ é escolhida.
2. Quando apenas uma solução é factível, ela é escolhida.
3. Quando as duas soluções são infactíveis, a que tiver o menor valor de *fitness*, calculado conforme a Equação (3.7), é escolhida.

Capítulo 4

Arquiteturas Propostas

Muitos dos trabalhos anteriores voltados à programação semafórica, dentre eles Oliveira (2009) e Costa (2012), utilizavam simuladores de tráfego para gerar os fluxos de veículos em cada movimento e também para medir os tempos gastos pelos veículos para percorrer o trecho sob estudo (atraso). Em geral, estes algoritmos seguem uma arquitetura similar a apresentada na Figura 4.1, que demanda a execução do simulador de tráfego milhares de vezes. Isso implica em horas (ou até dias) de execução para se obter boas soluções, tendo em vista o alto custo de execução do simulador.

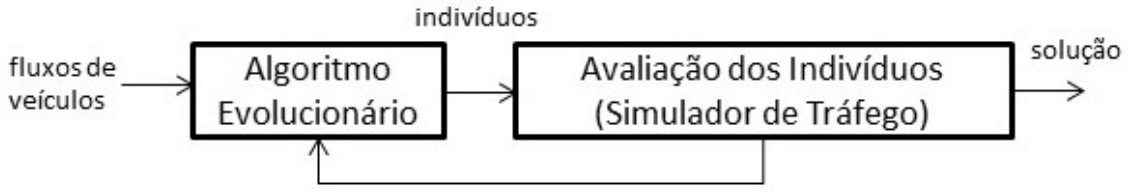


Figura 4.1: Arquitetura comum em trabalhos anteriores.

A primeira proposta deste trabalho consiste em alterar a arquitetura dos trabalhos anteriores para a apresentada na Figura 4.2. Nessa estrutura o simulado de tráfego é executado apenas no início do processo, com a finalidade de gerar cenários de fluxo de veículos nos movimentos a serem utilizados. A avaliação dos indivíduos, por sua vez, passa a ser feita com o uso de modelos matemáticos determinísticos. Dessa forma, a nova arquitetura possui um custo computacional muito menor, o que torna a otimização da programação semafórica muito mais rápida e possibilita seu uso em processos de otimização em tempo real.

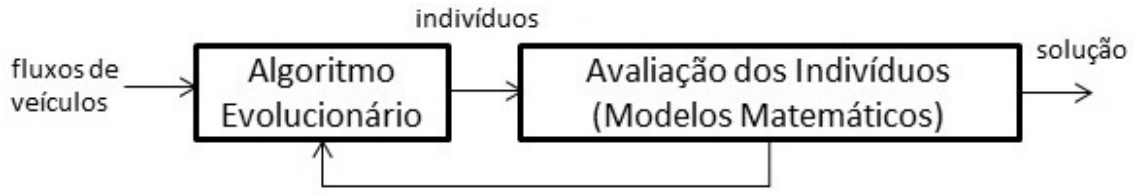


Figura 4.2: Arquitetura proposta para otimização da programação semafórica.

Porém, tendo em vista que as características de trânsito não mudam abruptamente ao longo do dia, realizar o processo de otimização completo nem sempre é necessário. Portanto, a segunda proposta deste trabalho é uma arquitetura de ajuste das programações semafóricas. Dadas programações semafóricas adequadas para determinados períodos do dia (obtidas, por exemplo, com a arquitetura apresentada na Figura 4.2), essas podem ser ajustadas para variações do cenário esperado ou para pequenos intervalos de tempo por meio de uma metaheurística de busca local, como proposto na Figura 4.3.

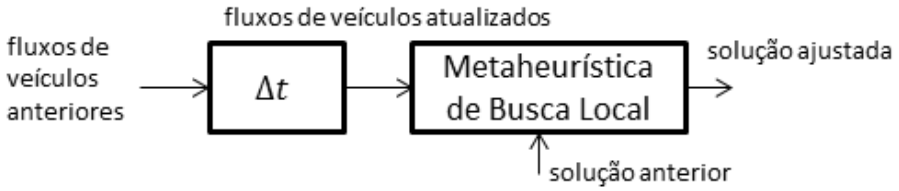


Figura 4.3: Arquitetura proposta para ajustes das programações semafóricas.

As novas arquiteturas possibilitam também avaliar os indivíduos para mais de um cenário de fluxo de veículos, de forma a se obter programações semafóricas robustas. Assim, espera-se que as soluções obtidas não se tornem ruins, caso o cenário real seja diferente do utilizado no processo de otimização. As ferramentas utilizadas e etapas mais importantes das duas propostas são discutidas no restante desse capítulo.

4.1 Simulador de Tráfego

Na primeira arquitetura proposta, o simulador de tráfego deve ser executado apenas no início do processo de otimização, com o intuito de gerar os fluxos de veículos nos movimentos. Com isso, foi desenvolvido um gerador de fluxos específico para este trabalho. Suas entradas são o fluxo de entrada de veículos no trecho sob análise, os movimentos possíveis ($\mathcal{M}$) e as chances dos veículos seguirem em cada movimento, que são dadas por estudos reais ou definidas pelo projetista de trânsito. O simulador sorteia os movimentos que cada um dos veículos

percorrerá, seguindo as probabilidades dadas, desde o movimento de entrada no trecho até o movimento de saída do mesmo e os contabiliza. Dessa forma, sua saída é o fluxo de veículos em todos os movimentos do trecho sob análise respeitando as suas restrições de balanceamento. O fluxograma desse processo é apresentado no Algoritmo 14.

Algoritmo 14 Gerador de Fluxos

```
1: function GeradorFluxos(M)
2:     for each m ∈ (movimentos de entrada ⊂ M) do
3:         for i ← 1 até m^c_fluxo do
4:             repeat
5:                 m ← SorteiaProximoMovimento(m_proximos, m_chances);
6:                 m^c_fluxo ← m^c_fluxo + 1;
7:             until m não seja movimentos de saída;
8:         end for
9:     end for
10:    return M;
11: end function
```

Vale destacar que este simulador pode ser substituído por qualquer outro disponível ou, ainda, por dados reais de contagem de veículos nos movimentos, caso estes estejam disponíveis.

4.2 Codificação das Soluções Candidatas

As soluções candidatas foram codificadas como vetores de números inteiros em todos os algoritmos propostos. Nessa representação, cada uma das $N_{posicoes}$ posições indica o tempo de verde do respectivo semáforo. Os ciclos, por sua vez, foram dispostos sequencialmente. A Figura 4.4 mostra um exemplo de indivíduo para um trecho com dois ciclos, cada um com quatro semáforos.

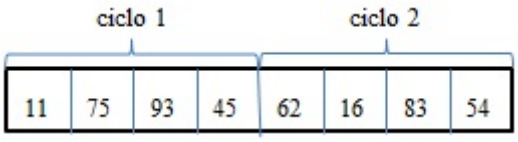


Figura 4.4: Exemplo de indivíduo com 2 ciclos, cada um com 4 semáforos.

A primeira população de todos os algoritmos evolucionários utilizados foi composta por indivíduos gerados aleatoriamente, onde os tempos de verde (t_{verde}) dos semáforos devem estar dentro do intervalo de verde permitido ($T_{verde\_min} \leq t_{verde} \leq T_{verde\_max}$). No entanto, dependendo do algoritmo, é possível que os tempos de verde dos semáforos (t_{verde}) ultrapassem os limites superior (T_{verde_max}) ou inferior (T_{verde_min}) nos processos de cruzamento ou mutação. Assim, foram definidos dois tipos de tratamentos possíveis:

Saturação: Os valores param nos limites superiores ou inferiores, conforme a Equação (4.1).

$$\begin{aligned} &(t_{verde} = T_{verde\_max} \text{ se } t_{verde} > T_{verde\_max}) \\ &\text{ ou} \\ &(t_{verde} = T_{verde\_min} \text{ se } t_{verde} < T_{verde\_min}) \end{aligned} \tag{4.1}$$

***Range* Cíclico:** Os valores passam para as extremidades contrárias da faixa de valores possíveis, conforme a Equação (4.2).

$$\begin{aligned} &(t_{verde} = T_{verde\_min} + u \text{ se } t_{verde} > T_{verde\_max} \text{ em } u) \\ &\text{ ou} \\ &(t_{verde} = T_{verde\_max} - u \text{ se } t_{verde} < T_{verde\_min} \text{ em } u) \end{aligned} \tag{4.2}$$

4.3 Restrições do Problema

O problema de otimização da programação semafórica possui duas restrições básicas a serem consideradas: os tempos mínimo e máximo de ciclo ($T_{ciclo\_min} \leq t_{ciclo} \leq T_{ciclo\_max}$). O tempo mínimo de clico é garantido pela definição dos tempos mínimos de verde (T_{verde_min}). Já a restrição do tempo máximo de ciclo foi tratada por meio de penalidade. As soluções que possuem ciclos com tempo superiores ao limite ($t_{ciclo} > T_{ciclo\_max}$) sofrem uma penalização na função objetivo, conforme estabelecido na Equação (3.7). A penalidade, por sua vez, foi definida como sendo uma taxa ($T_{penalidade}$), dada em segundos, acrescentada para cada ponto percentual em que o tempo de algum ciclo estiver acima do limite superior T_{ciclo_max}. Seu cálculo é exibido na Equação (4.3).

$$g(x) = \sum_{c \in x_{\mathcal{C}}} \max \left[\left(\frac{t^c_{ciclo} - T_{ciclo\_max}}{T_{ciclo\_max}} \right) \times 100,0 \right] \times T_{penalidade} \tag{4.3}$$

onde:

g é a penalidade a ser aplicada ao indivíduo (s);

$x_{\mathcal{C}}$ é o conjunto de ciclos presentes na solução candidata x;

t^{c}_{ciclo} é o tempo de ciclo do ciclo c (s);

T_{ciclo_max} é o tempo de ciclo máximo (s);

$T_{penalidade}$ é a taxa de penalidade definida pelo projetista (s).

Essa estratégia foi definida devido à três fatores:

- programações que possuam ciclos com tempos superiores ao limite máximo ainda podem gerar atrasos inferiores às demais e, com isso, serem interessantes;
- os algoritmos evolucionários têm grandes probabilidade de gerar soluções infactíveis e ajustá-las ou substituí-las prejudicaria o seu tempo de execução;
- caso seja necessário, o projetista de trânsito pode eliminar as soluções infactíveis apenas escolhendo uma taxa de penalidade ($T_{penalidade}$) muita alta.

4.4 Teste de Robustez

Na maioria dos algoritmos de otimização, assume-se que os dados de entrada são determinísticos. Porém, em muitos problemas práticos, podem existir erros de medições ou variações nos dados de entrada, fazendo que soluções deixem de ser ótimas ou se tornem infactíveis em cenários distintos do considerado durante a otimização (Ben-Tal and Nemirovski, 2000).

O problema de otimização semafórica é um bom exemplo de onde esses problemas acontecem, visto que o processo de contagem de veículos nos trechos realizados pelas empresas de trânsito pode ser impreciso e que os fluxos de veículos variam significativamente ao longo do dia ou no mesmo horário, em dias diferentes da semana. Assim, apenas um cenário de fluxo de veículos nos movimentos é, em geral, insuficiente para simular situações reais.

Com isso, surgiu a necessidade de se avaliar as soluções candidatas obtidas pelos algoritmos utilizados neste trabalho para um conjunto ($\mathcal{S}$) de cenários de fluxos de veículos nos movimentos (entradas vizinhas), de forma que os mesmos pudessem simular as variações que ocorrem em situações reais. Esses cenários foram obtidos em dois passos:

1. A partir do fluxo de entrada de veículos original foram gerados $N_{cenarios}$ cenários de fluxo de entradas de veículos alternativos no trecho sob análise. Esses cenários foram gerados utilizando uma distribuição de probabilidade Poisson Haight (1967).
2. A partir dos $N_{cenarios}$ cenários de fluxos de entradas de veículos no trecho foi gerado, com o uso do simulador desenvolvido, o conjunto $\mathcal{S}$ de cenários de fluxos de veículos em cada um dos movimentos.

Por fim, o conjunto $\mathcal{S}$ de cenários de fluxos de veículos nos movimentos foi utilizado na construção dos modelos matemáticos determinísticos presentes nas funções objetivos a serem tratadas no processo de otimização.

4.5 Avaliação dos Indivíduos

A partir dos modelos matemáticos, apresentados na Seção 2.4, e dos conceitos de robustez, vistos na Seção 4.4, foram determinadas as funções objetivo utilizadas neste trabalho para avaliação dos indivíduos nos problemas mono e multiobjetivo.

Problema Mono-objetivo

- Otimizar o caso médio das avaliações dos atrasos totais dos veículos no trecho, com base no conjunto $\mathcal{S}$ de cenários de fluxos de veículos nos movimentos – Equação (4.4).

$$f_{caso\_medio}(x) = \frac{\sum_{s \in \mathcal{S}} t(x,s)}{|\mathcal{S}|} + g(x) \tag{4.4}$$

onde:
$\mathcal{S}$ é o conjunto de cenários de fluxos de veículos nos movimentos;
$t(x,s)$ é o total dos veículos no trecho (s) calculado para o indivíduo x ao considerar o cenário de fluxo de veículos nos movimentos s, conforme Equação (2.4);
$g(x)$ é a penalidade de função objetivo do indivíduo x, conforme Equação (4.3).

O objetivo por trás dessa função é obter as programações semafóricas que permitam aos veículos, na média, passarem pelo trecho com os menores atrasos.

Problema Multiobjetivo

- Otimizar o caso médio das avaliações dos atrasos totais – Equação (4.4);
- Minimizar o pior caso dos atrasos totais – Equação (4.5).

$$f_{pior\_caso}(x) = \max_{s \in \mathcal{S}} t(x, s) + g(x) \tag{4.5}$$

Com esse modelo espera-se obter programações semafóricas que permaneçam adequadas mesmo em cenários de fluxos que gerem atrasos consideravelmente superiores ao caso médio.

4.6 Algoritmos Utilizados

Três algoritmos evolucionários foram aplicados ao problema mono-objetivo: AG, DE e PSO. Nos três casos a condição de parada foi o número máximo de iterações ($N_{iteracoes}$) e as soluções obtidas foram utilizadas como soluções iniciais do VNS, para fins de refinamento. Já no caso multiobjetivo, a ferramenta de otimização utilizada foi o algoritmo NSGA-II. Detalhes da implementação de todos os métodos utilizados são apresentados na sequência.

4.6.1 Algoritmo Genético

No AG desenvolvido, a seleção é realizada com o *Stochastic Universal Sampling* (SUS) (Baker, 1987) e elitismo.

O cruzamento pode ser feito por dois operadores: cruzamento de dois pontos (*two points crossover*) e cruzamento uniforme (*uniform crossover*) (Goldberg, 1989). A escolha entre eles ocorre de forma aleatória, com base em uma distribuição uniforme.

A mutação utiliza uma variação do operador *Flip*. Para cada posição a ser alterada é escolhido um novo valor inteiro aleatório no intervalo $[v_{atual} - \Delta; v_{atual} + \Delta]$, onde v_{atual} é o valor atual da posição e Δ é a máxima perturbação admissível para a mutação. Caso necessário, os intervalos saturam no limite inferior (T_{verde_min}) ou no limite superior (T_{verde_max}) da variável.

4.6.2 *Particle Swarm Otmization*

O PSO foi o segundo algoritmo evolucionário mono-objetivo utilizado neste trabalho e sua implementação seguiu a apresentada na Seção 3.1.2. Os limites da variável são tratados no processo de mutação por meio de *range* cíclico.

4.6.3 *Differential Evolution*

O DE foi implementado conforme apresentado na Seção 3.1.3. Para mutação diferencial foi escolhido o operador *current-to-best*, após testes empíricos contra os operadores *random-to-random* e *current-to-random*. Os limites das variáveis também foram tratados por *range* cíclico.

4.6.4 *Non-dominated Sorting Genetic Algorithm* II

O NSGA-II foi o algoritmo utilizado neste trabalho para a variante multiobjetivo do problema. Sua implementação foi feita conforme apresentado na Seção 3.2.1. Os operadores de cruzamento e mutação foram os mesmos adotados no AG (Seção 4.6.1).

4.6.5 *Variable Neighboorhood Search*

Por último, o VNS foi a metaheurística de busca local escolhida neste trabalho para refinar as soluções obtidas pelos algoritmos evolucionários e para realizar os ajustes necessários nas programações semafóricas. Assim, ele foi implementado como descrito na Seção 3.3.1 e utilizou o método *First Improvement* para as buscas locais.

As vizinhanças, por sua vez, foram definidas como a adição de um valor Δ^i, que varia de $+\Delta_{max}$ a $-\Delta_{max}$, ao tempo de verde em cada uma das $N_{posicoes}$ posições do indivíduo. A ordem das vizinhanças foi escolhida após testes empíricos, onde foi detectada a tendência do problema em obter melhores soluções para tempos mais altos de verde. Caso necessário o algoritmo trata os limites das variáveis como *range* cíclico. A Figura 4.5 mostra um exemplo de como o processo funciona, tendo um indivíduo de 3 posições e $\Delta_{max} = 1$.

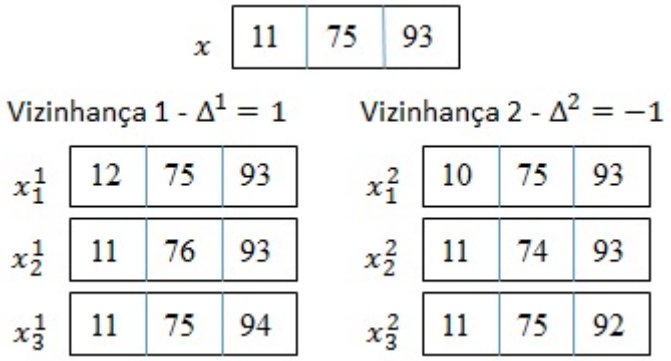


Figura 4.5: Exemplo de como o processo de mudança de vizinhanças do VNS funciona, tendo um indivíduo de 3 posições e $\Delta_{max} = 1$.

É possível notar que o número de soluções candidatas dentro de cada vizinhança é $N_{posicoes}$, enquanto o número de vizinhanças é $2\Delta_{max}$.

Capítulo 5

Experimentos e Resultados

Conforme dito na Seção 1.2, o objetivo deste trabalho é propor duas arquiteturas (apresentadas no Capítulo 4) para se obter programações semafóricas otimizadas e robustas em tempo real. Dessa forma, surgiram perguntas relevantes:

- Qual é o melhor algoritmo evolucionário para o problema tratado?
- Uma metaheurística de busca local pode ser útil para o problema?
- Permitir que existam ciclos no trecho com tempos superiores ao limite máximo estabelecido pode ser interessante em situações reais?
- A variação do fluxo de veículos no trecho realmente prejudica as programações semafóricas utilizadas? Se sim, como esses impactos podem ser reduzidos?
- Utilizar uma arquitetura de otimização reativa, que considera pequenas variações no fluxo de veículos, é mesmo viável para otimização em tempo real?

Para responder essas perguntas foram realizados quatro experimentos:

Comparação dos algoritmos evolucionários: os três AEs foram comparados, a fim de identificar o que melhor se adapta ao problema tratado. Também, foram avaliado os benefícios do VNS para refinar as soluções obtidas.

Testes das penalidades nas soluções: foi estudado o impacto do valor da taxa de penalidade na solução final obtida, especialmente quanto ao tamanho do ciclo.

Testes de robustez: cenários alternativos de fluxos de veículos nos movimentos foram avaliados com o intuito de estimar o impacto da variação do fluxo na qualidade final da solução obtida.

Teste dos ajustes das programações semafóricas: por fim, a segunda arquitetura proposta neste trabalho foi avaliada como uma alternativa para correção reativa da programação semafórica em tempo real, tendo em vista as alterações de tráfego ocorridas ao longo do tempo.

Os dados sobre os experimentos também são apresentados na sequência.

5.1 Dados dos Experimentos

Nesta seção são apresentados os trechos reais utilizados, assim como os ambientes e os parâmetros dos experimentos.

5.1.1 Trechos Escolhidos

Para realizar os testes das arquiteturas propostas neste trabalhos, foram escolhidos três trechos reais da cidade de Belo Horizonte:

- a Praça Raul Soares;
- a Avenida Amazonas, entre as praças Sete e Raul Soares;
- a Avenida Amazonas, entre a Praça Sete a Avenida do Contorno.

Cada uma dessas instâncias é discutida na sequência.

Praça Raul Soares

O primeiro trecho escolhido foi a Praça Raul Soares (Figura 5.1), onde os semáforos possuem dependências cíclicas. Assim, o parâmetro de qualidade dos movimentos deste trecho foi $J_m = 2,4$, conforme sugerido em Akçelik (1991). Já o fluxo de saturação de todos os movimentos foi definido como $Q = 1800V/s$, seguindo orientação do TRB (2010). Os dados deste trecho são apresentados na Tabela 5.1.

Tabela 5.1: Informações sobre o Trecho Praça Raul Soares.

Número de Semáforos	12
Número de Ciclos	5
Número de Movimentos	20

Figura 5.1: Trecho Praça Raul Soares (extraído do Google Maps).

Avenida Amazonas I - Entre Praça Sete e Praça Raul Soares

O segundo trecho utilizado foi um segmento da Avenida Amazonas, entre as praças Sete e Raul Soares (Figura 5.2). O mesmo foi escolhido por ser uma avenida/corredor onde os semáforos devem ser sincronizados. Dessa forma, o parâmetro de qualidade de seus movimentos foi definido como $J_m = 1,2$, seguindo Akçelik (1991). Mais uma vez, o fluxo de saturação de todos os movimentos foi $Q = 1800V/s$. Os dados da rede são descritos na Tabela 5.2.

Tabela 5.2: Informações sobre o Trecho Avenida Amazonas I.

Número de Semáforos	19
Número de Ciclos	6
Número de Movimentos	42

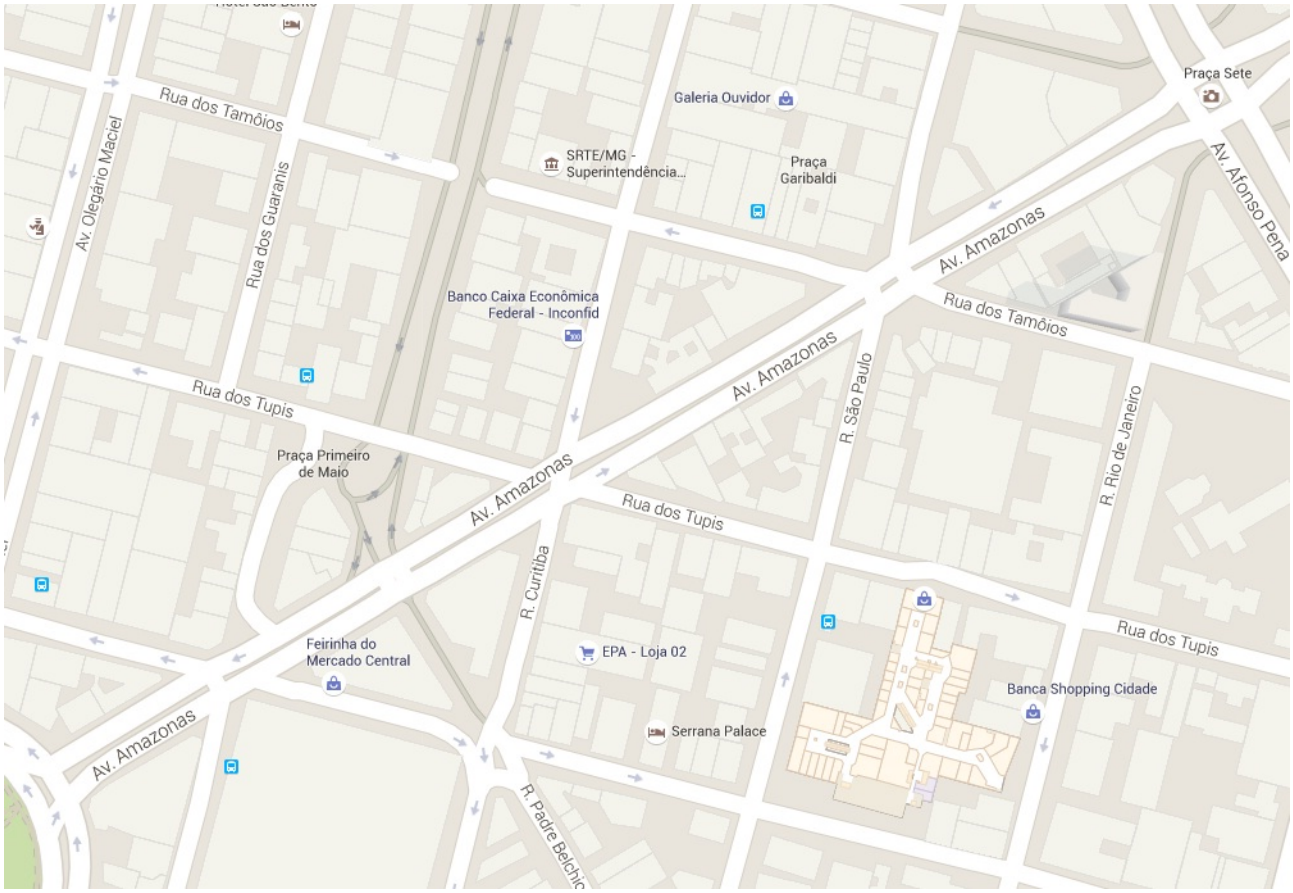


Figura 5.2: Trecho Avenida Amazonas I - Entre as praças Sete e Raul Soares (extraído do Google Maps).

Avenida Amazonas II - Praça Sete a Avenida do Contorno

Por fim, o terceiro trecho selecionado foi um segmento mais longo da Avenida Amazonas, entre a Praça Sete e a Avenida do Contorno, incluindo a Praça Raul Soares (Figura 5.3). Esse trecho é um importante corredor da cidade de Belo Horizonte e possui um número significativo de semáforos e ciclos (vide Tabela 5.3). Seus parâmetros foram os mesmos do trecho Avenida Amazonas I ($J_m = 1,2$ e $Q = 1800V/s$).

Tabela 5.3: Informações sobre o Trecho Avenida Amazonas II.

No. Semáforos	56
No. Ciclos	18
No. Movimentos	119

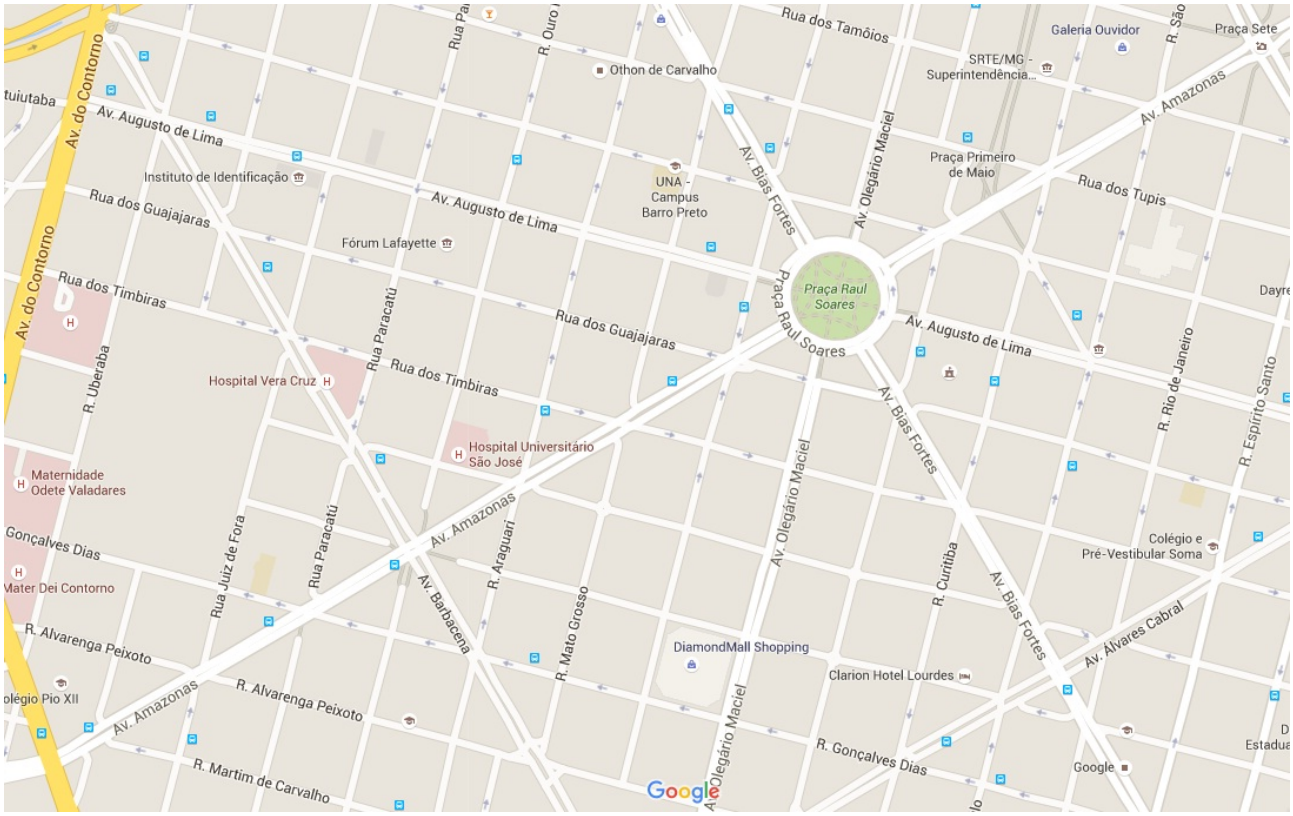


Figura 5.3: Trecho Avenida Amazonas II - Praça Sete a Avenida do Contorno (extraído do Google Maps).

5.1.2 Ambiente de Testes

Os experimentos realizados neste trabalhos foram feitos em dois computadores:

1. Wokstation HP Z220 com processador Intel Xeon Quad Core, 32GB de memória RAM e sistema operacional Windows 10 (64 bits);

2. PC Lenovo com processador Intel Core I3-2100 (3,1 GHz), 2,85 GB de memória RAM e sistema operacional Windows 7 (32 bits).

Os algoritmos foram implementados de forma serial, em Python 3.4.2.

5.1.3 Parâmetros dos Experimentos

Os parâmetros utilizados nos experimentos, salvo quando houver indicação contrária, são apresentados nesta seção.

A Tabela 5.4 exibe os parâmetros utilizados na construção das funções objetivo dos problemas mono e multiobjetivo (vide Equações 4.4 e 4.5)[1].

Tabela 5.4: Parâmetros utilizados na construção das funções objetivo dos problemas mono e multiobjetivos.

Parâmetro	Valor
$T_{penalidade}$	10s
T_{ciclo_max}	220s
T_{verde_min}	10s
T_{verde_max}	100s
t_m^0	0s
T_f	3600s

A Tabela 5.5 descreve os parâmetros de execução dos algoritmos utilizados neste trabalho, definidos após testes preliminares. Os AEs também tiveram como critério de parada o total de 1000 iterações e suas populações foram compostas por 100 indivíduos.

Tabela 5.5: Parâmetros de execução dos algoritmos nos experimentos realizados.

Parâmetro	Valor
AG e NSGA-II - p_{cross}	1
AG e NSGA-II - p_{mut}	0,2
AG e NSGA-II - Δ	5s
PSO - p_{mut}	0,3
PSO - ω	0,2
PSO - ϕ_l	0,5
PSO - ϕ_g	0,5
DE - p_{mut}	1
DE - p_{cross}	1
DE - μ	0,3
VNS - k_max	5
VNS - Δ	5s

[1] Os valores referentes ao tempo máximo de ciclo (T_{ciclo_max}), tempo mínimo de verde (T_{verde_min}) e tempo máximo de verde (T_{ciclo_max}) foram extraídos do trabalho de Costa (2012).

5.2 Comparação dos Algoritmos Evolucionários

Nesta seção são aprestados os resultados dos testes comparativos entre os três AEs de otimização mono-objetivo utilizados neste trabalho. Os algoritmos foram avaliados em relação a três critérios:

Qualidade das soluções: neste critério busca-se o algoritmo que apresente as melhores soluções, ou seja, as programações semafóricas que permitam aos veículos percorrerem o trecho com o menor atraso possível;

Convergência das soluções: este critério avalia a média e o desvio padrão das soluções obtidas, assim como o número de iterações necessárias para que os algoritmos possam obter soluções aproximadas;

Tempo de execução: este critério verifica o tempo de execução dos algoritmos.

Por fim, também foi analisado o ganho de se utilizar o VNS como mecanismo de refinamento das soluções obtidas pelos AEs.

Neste experimento os algoritmos foram executados 33 vezes, na Wokstation HP Z220, para os trechos Praça Raul Soares e Avenida Amazonas I (descritos na Seção 5.1.1) e seus indivíduos foram avaliadas com uso do modelo matemático de Akçelik (1991) para 10 cenários de fluxos de veículos nos movimentos.

5.2.1 Trecho Praça Raul Soares

Os resultados obtidos para cada um dos algoritmos para o trecho Praça Praça Raul Soares são apresentados nas Tabelas 5.6, 5.7 e 5.8. A comparação dos mesmos pode ser vista na Figura 5.4.

Tabela 5.6: Resultados do PSO + VNS para o trecho Praça Raul Soares.

	PSO - Atrasos (s)		
Resultado	Algoritmo	VNS	% de melhoria do VNS
Melhor	16140	16140	0,00
Pior	16140	16140	0,00
Média	16140	16140	0,00
Mediana	16140	16140	
Desvio Padrão	0,00	0,00	0,00

Tabela 5.7: Resultados do AG + VNS para o trecho Praça Raul Soares.

	AG - Atrasos (s)		
Resultado	Algoritmo	VNS	% de melhoria do VNS
Melhor	16142	16140	0,01
Pior	16155	16148	0,04
Média	16149	16146	0,03
Mediana	16149	16144	
Desvio Padrão	3,71	1,87	49,60

Tabela 5.8: Resultados do DE + VNS para o trecho Praça Raul Soares.

	DE - Atrasos (s)		
Resultado	Algoritmo	VNS	% de melhoria do VNS
Melhor	16171	16141	0,19
Pior	16331	16194	0,84
Média	16227	16147	0,49
Mediana	16217	16144	
Desvio Padrão	36,81	9,72	73,59

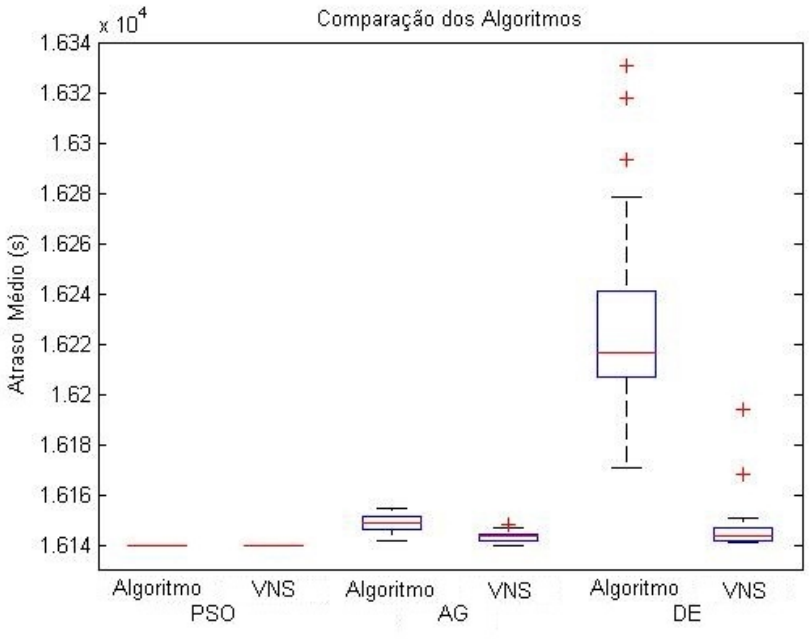


Figura 5.4: Comparação dos resultados dos algoritmos evolucionários para o trecho Praça Raul Soares.

A Tabela 5.9 apresenta o número de iterações de cada um dos algoritmos[2], assim como seus tempos de execução. A Figura 5.5 exibe a convergência dos algoritmos.

Tabela 5.9: Tempos de execução dos métodos para o trecho Praça Raul Soares.

	PSO		AG		DE	
	Algoritmo	VNS	Algoritmo	VNS	Algoritmo	VNS
Número médio de iterações	1000	5,00	1000	8,15	1000	24,88
Tempo médio por iteração (s)	0,248	0,207	0,350	0,196	0,349	0,116

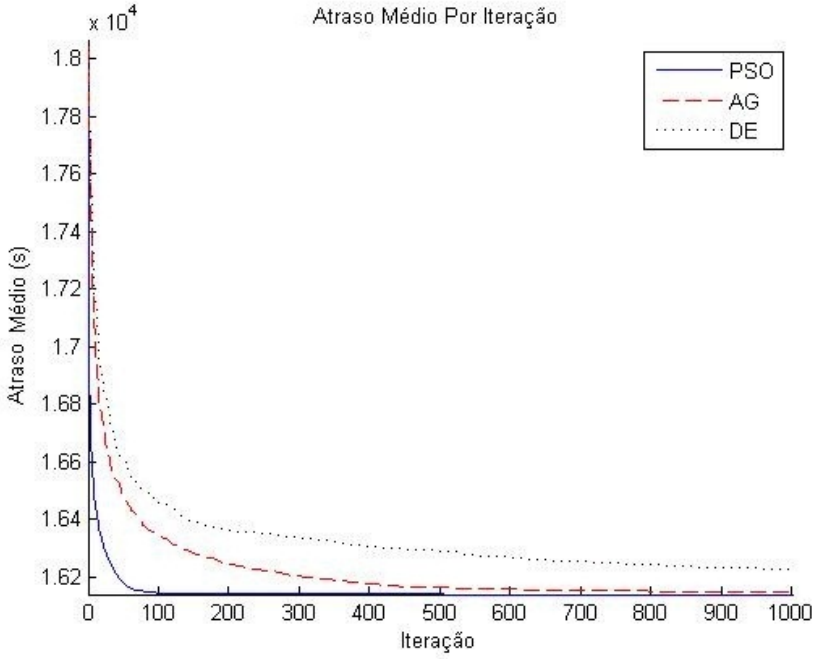


Figura 5.5: Curva de convergência média dos algoritmos evolucionários para o trecho Praça Raul Soares.

A partir dos experimentos realizados foi possível perceber que o PSO foi o AE que obteve a melhor solução (menor atraso) e convergiu para ela em todas as 33 execuções. Além disso, ele foi o que convergiu com o menor número de iterações (Figura 5.5) e apresentou o menor tempo de execução (Tabela 5.9).

O uso VNS para refinar as soluções também se mostrou interessante. Apesar de não trazer melhorias significativas para as melhores soluções, ao se analisar todas as execuções é possível perceber a diminuição do desvio padrão das soluções obtidas pelos algoritmos AG e DE (Tabelas 5.7 e 5.8). Dessa forma, ele contribuiu para reduzir a variabilidade da resposta dos algoritmos.

[2]Cada iteração dos AEs representa a avaliação dos 100 indivíduos presentes em suas populações. A iteração do VNS, por sua vez, representa um *First Improvement*, sendo no máximo $2\Delta_{max}N_{posicoes}$ avaliações.

5.2.2 Trecho Avenida Amazonas I

Os resultados dos algoritmos para o trecho Avenida Amazonas I são descritos nas Tabelas 5.10, 5.11 e 5.12. A Figura 5.6, por sua vez, mostra o resumo dos mesmos.

Tabela 5.10: Resultados do PSO + VNS para o trecho Avenida Amazonas I.

	PSO - Atrasos (s)		
Resultado	Algoritmo	VNS	% de melhoria do VNS
Melhor	21110	21103	0,03
Pior	21137	21129	0,04
Média	21119	21112	0,03
Mediana	21118	21112	
Desvio Padrão	5,43	4,62	14,92

Tabela 5.11: Resultados do AG + VNS para o trecho Avenida Amazonas I.

	AG - Atrasos (s)		
Resultado	Algoritmo	VNS	% de melhoria do VNS
Melhor	21195	21112	0,39
Pior	21388	21225	0,76
Média	21288	21145	0,67
Mediana	21276	21134	
Desvio Padrão	47,44	25,99	45,22

Tabela 5.12: Resultados do DE + VNS para o trecho Avenida Amazonas I.

	DE - Atrasos (s)		
Resultado	Algoritmo	VNS	% de melhoria do VNS
Melhor	21450	21111	1,58
Pior	22410	21299	4,96
Média	21724	21169	2,56
Mediana	21652	21163	
Desvio Padrão	232,74	46,67	79,95

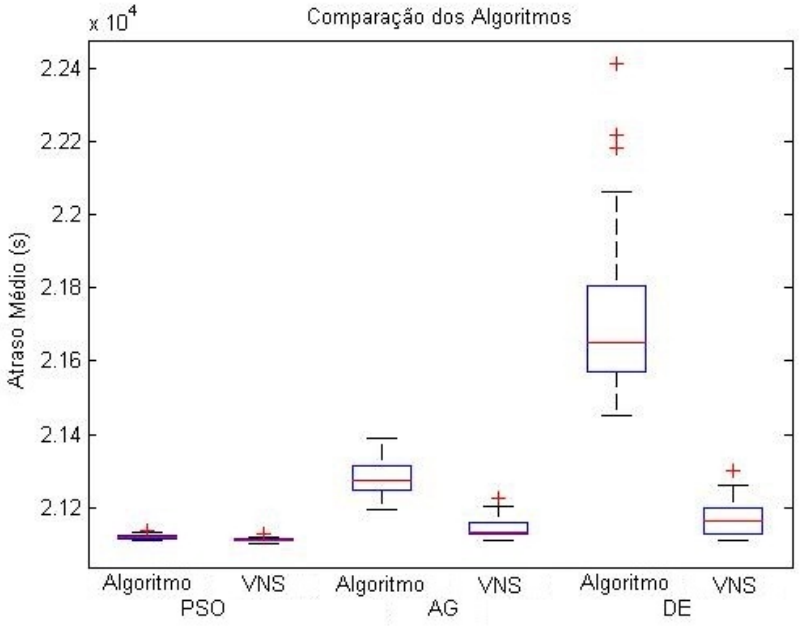


Figura 5.6: Comparação dos resultados dos algoritmos evolucionários para o trecho Avenida Amazonas I.

O tempo de execução das iterações dos algoritmos podem ser vistos na Tabela 5.13, enquanto a Figura 5.7 apresenta a convergência dos algoritmos.

Tabela 5.13: Tempos de execução dos métodos para o trecho Avenida Amazonas I.

	PSO		AG		DE	
	Algoritmo	VNS	Algoritmo	VNS	Algoritmo	VNS
Número médio de iterações	1000	8,39	1000	39,67	1000	87,79
Tempo médio por iteração (s)	0,474	0,486	0,634	0,340	0,654	0,297

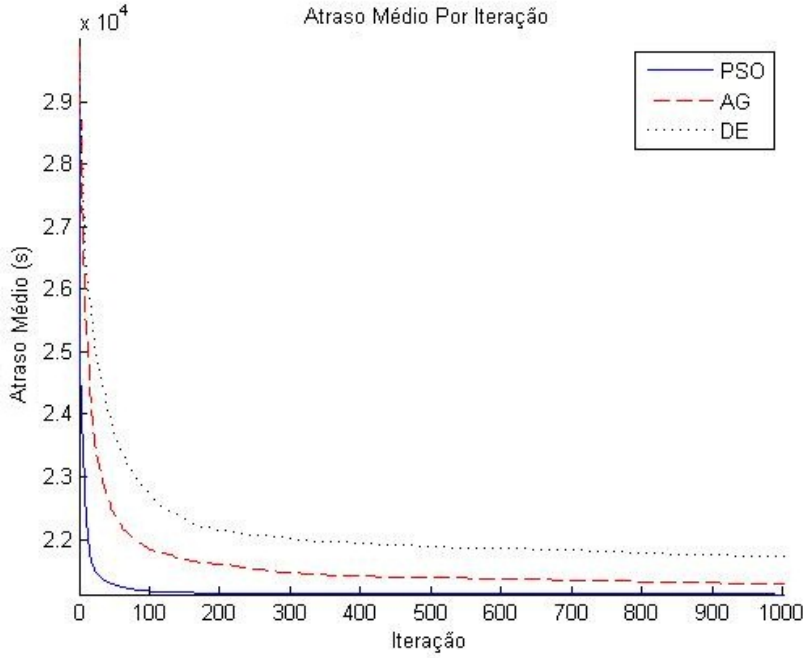


Figura 5.7: Curva de convergência média dos algoritmos evolucionários para o trecho Avenida Amazonas I.

Assim como nos resultados obtidos para o trecho Praça Raul Soares, o PSO voltou a ser o melhor algoritmo para o trecho Avenida Amazonas I nos três quesitos analisados e, portanto, sugere-se seu uso para resolução do problema da otimização da programação semafórica tratado neste trabalho. Também foi possível observar grande redução no desvio padrão das soluções após o refinamento realizado pelo VNS, principalmente em relação aos resultados obtidos pelo DE, conforme apresentados na Tabela 5.12.

5.2.3 Considerações Finais

A partir dos resultados apresentados nas Tabelas 5.9 e 5.13 foi possível perceber que os tempos de execução do processo de otimização da nova arquitetura (Figura 4.2) foram consideravelmente inferiores aos observados na arquitetura baseada em simulação (Figura 4.1), utilizada nos trabalhos de Oliveira (2009) e Costa (2012).

Naqueles trabalhos, o simulador gastava aproximadamente 0,2s para avaliar cada um dos indivíduo para apenas um cenário de fluxo de veículos nos movimentos (Gonzaga et al., 2015). Com isso, os métodos demoravam horas para retornar boas soluções, visto que realizavam 1000 iterações e utilizavam uma população de 100 indivíduos. Já com o uso da nova arquitetura,

foi possível executar iterações inteiras (avaliar os 100 indivíduos para 10 cenários de fluxos de veículos) em tempos próximos de 0,5s. Assim, os tempos de execução dos algoritmos passaram a ser de apenas alguns minutos. Vale ressaltar que poderiam ser utilizadas estratégias para torná-los ainda mais rápidos, a saber:

- encerrá-los após certo número de iterações sem melhorias significativas, uma vez que não dependeram das 1000 iterações para obter soluções aproximadas (vide Figuras 5.5 e 5.7);
- implementá-los em uma linguagem compilada, como C ou C++, visto o *overhead* do Python, por ser uma linguagem interpretada;
- desenvolvê-los em um ambiente de programação paralela, de forma que as avaliações dos indivíduos fossem realizadas simultaneamente.

Dessa forma, abre-se a possibilidade para que essa nova arquitetura seja utilizada para a otimização das programações semafóricas em tempo real.

5.3 Testes de Penalidade das Soluções

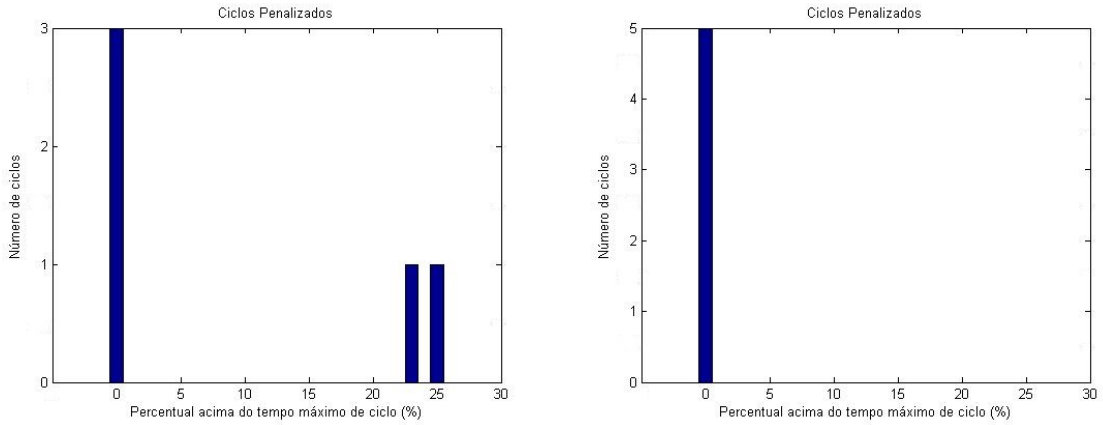
Nesta seção é apresentado como a taxa de penalidade influencia no processo de otimização e, consequentemente, nas programações semafóricas obtidas. Para este teste, o algoritmo PSO + VNS foi executados 33 vezes para cada trecho e foram consideradas três possíveis taxas de penalidade ($T_{penalidade} = 0s, 10s$ ou $1000s$). Assim como no Teste de Comparação dos Algoritmos (Seção 5.2), as soluções candidatas também foram avaliadas com uso do modelo de Akçelik (1991) para 10 cenários de fluxos de veículos nos movimentos.

5.3.1 Trecho Praça Raul Soares

A Tabela 5.14 apresenta os resultados dos testes de penalidade para o trecho Praça Raul Soares. A Figura 5.8, por sua vez, exibe o comportamento dos ciclos nas melhores programações semafóricas obtidas.

Tabela 5.14: Resultados do teste de penalidade para o trecho Praça Raul Soares.

$T_{penalidade}$	0s	10s
Melhor Solução - Atraso (S)	15938	16140
Número de Ciclos Penalizados	2	0



(a) $T_{penalidade} = 0s$ (b) $T_{penalidade} = 10s$

Figura 5.8: Ciclos que sofreram penalidades no trecho Praça Raul Soares.

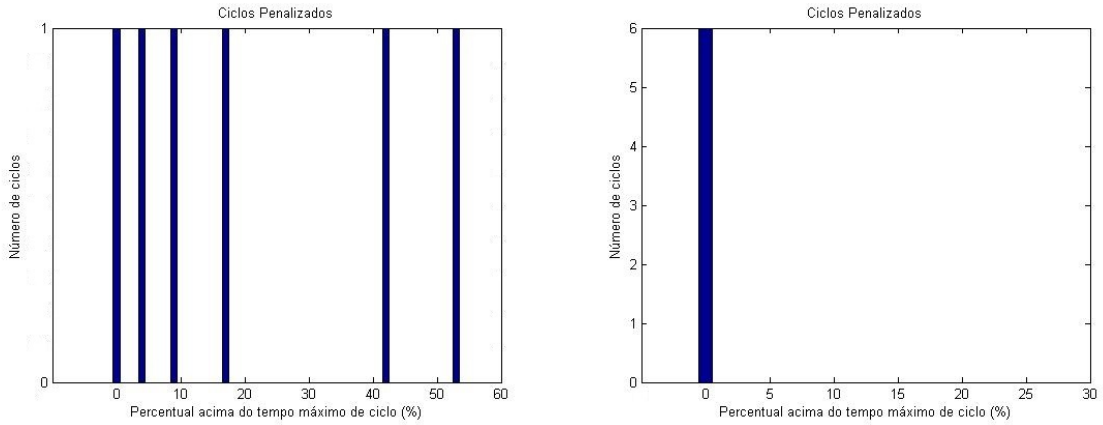
A partir dos resultados descritos na Tabela 5.14 é possível perceber que as programações semafóricas que consideram o uso de ciclos com tempos superiores ao limite máximo definido (T_{ciclo_max}) podem resultar em atrasos menores quando comparadas às soluções que respeitam a restrição (vide Tabela 5.14). Porém, não definir uma taxa de penalidade ($T_{penalidade} = 0s$) significa que o processo de otimização pode definir livremente os tempos. Isso pode gerar alguns tempos de ciclos muito longos, o que prejudica o fluxo dos veículos nos demais movimentos do trecho. Na Figura 5.8a, por exemplo, é possível perceber que 2 dos 5 ciclos tiveram tempos mais de 20% acima do limite superior.

5.3.2 Trecho Avenida Amazonas I

Os resultados dos testes para o trechos Avenida Amazonas I são descritos na Tabela 5.14 e o comportamento dos ciclos são apresentados na Figura 5.9.

Tabela 5.15: Resultados do teste de penalidade para o trecho Avenida Amazonas I.

$T_{penalidade}$	0s	10s
Melhor Solução - Atraso (s)	20438	21103
Número de Ciclos Penalizados	5	0



(a) $T_{penalidade} = 0s$ (b) $T_{penalidade} = 10s$

Figura 5.9: Ciclos que sofreram penalidades no trecho Avenida Amazonas I.

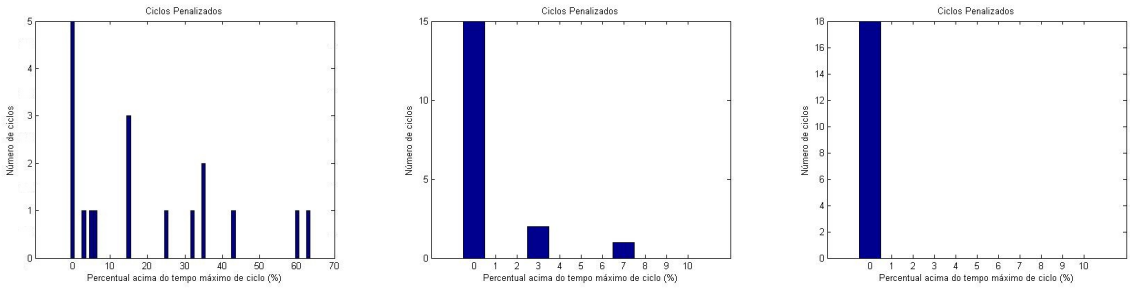
Mais uma vez as soluções que consideram o uso de ciclos com tempos superiores ao tempo máximo definido ($t_{ciclo} > T_{ciclo\_max}$) apresentaram atrasos inferiores às soluções que respeitaram a restrição (Tabela 5.15). Contudo, para este trecho, a maioria dos ciclos (5/6) apresentaram tempos superiores ao limite e a metade (3/6) ultrapassou o limite em 10% ou mais.

5.3.3 Trecho Avenida Amazonas II

A Tabela 5.14 exibe os resultados dos experimentos para o trecho Avenida Amazonas II. A Figura 5.10, por sua vez, apresenta o comportamento dos ciclos nas melhores soluções.

Tabela 5.16: Resultados do teste de penalidade para o trecho Avenida Amazonas II.

$T_{penalidade}$	0s	10s	1000s
Melhor Solução - Atraso (s)	77167	79429	79512
Número de Ciclos Penalizados	13	3	0



(a) $T_{penalidade} = 0s$ (b) $T_{penalidade} = 10s$ (c) $T_{penalidade} = 1000s$

Figura 5.10: Ciclos que sofreram penalidades no trecho Avenida Amazonas II.

Os resultados vistos na Tabela 5.16 e na Figura 5.10a, deixam claro como os tempos dos ciclos se comportam a medida em que a taxa de penalidade aplicada às soluções aumenta. Novamente a melhor programação semafórica obtida foi quando a penalidade não foi aplicada ($T_{penalidade} = 0$). No entanto, a maior parte dos ciclos dessa solução tiveram tempos superiores ao limite máximo (13/18), sendo que alguns (2/18) ultrapassaram 50%.

O uso de uma taxa de penalidade média ($T_{penalidade} = 10$) levou a um comportamento interessante. O atraso de sua solução foi inferior ao da solução obtida quando foi utilizada uma taxa de penalidade restritiva ($T_{penalidade} = 1000$). Porém, apenas os ciclos que realmente precisavam ser privilegiados tiveram tempos acima do limite. Além disso, os tempo em excesso nunca ultrapassaram 10%.

5.3.4 Considerações Finais

Os testes realizados demonstraram que penalizar as soluções que consideram ciclos com tempos superiores ao limite foi melhor que simplesmente descartá-las. Além de facilitar o desenvolvimento dos algoritmos de otimização e torná-los mais rápidos, suas programações demonstração serem boas opções para uso.

Porém, os projetistas de tráfego devem escolher a taxa de penalidade a ser utilizada com cuidado. Quando esta é muito alta, movimentos ou ciclos que possuem grandes fluxos de veículos podem ser prejudicados, pois ficam limitados. Isso pode aumentar o atraso do trecho e, em alguns casos, não permitir a dispersão das filas, causando engarrafamentos. Por outro lado, taxas de penalidades muito baixas podem gerar tempos de ciclos muito altos, como os visto nas Figuras 5.9a e 5.10a. Nesse caso, pode ocorrer grande variância das velocidades médias dos veículos no trecho, uma vez que muitos deles podem ficar grandes períodos de tempos parados nos semáforos.

Assim, a taxa de penalidade deve permitir tempos um pouco superiores ao limite para alguns ciclos que possuam grandes fluxos de veículos, de forma a priorizá-los. Contudo, é importante que ela também seja capaz de garantir o equilíbrio do trecho como um todo.

Os detalhes das programações avaliadas nesta seção podem ser vistos no Apêndice A.

5.4 Testes de Robustez

Como visto na Seção 4.4, a melhor programação obtida para um cenário de fluxo de veículos não é necessariamente a melhor programação a ser utilizada na prática, pois os fluxos podem variar consideravelmente.

Assim, nesta seção são apresentados experimentos onde as soluções candidatas são avaliadas para vários cenários de fluxo de veículos alternativos. Com essa avaliação, é possível estimar a influencia das variações dos fluxos nas programações semafóricas obtidas. Para tal foram utilizadas variantes mono-objetivo e multiobjetivo do problema.

5.4.1 Testes Mono-objetivo

O primeiro teste de robustez foi feito para o problema de programação semafórica mono-objetivo, onde espera-se obter o menor atraso médio tendo em vista o conjunto de cenários gerado. O algoritmo PSO + VNS foi executados 33 vezes[3] para cada trecho e foram utilizados conjuntos de cenários de fluxos de veículos nos movimentos com diferentes tamanhos: 1 e 10 cenários para Praça Raul Soares e Avenida Amazonas I e 1, 10 e 50 cenários para Avenida Amazonas II.

Trecho Praça Raul Soares

As Tabelas 5.17 e 5.18 apresentam os resultados e tempos de execução a partir do modelo matemático de Akçelik (1991) para o teste de robustez no trecho Praça Raul Soares.

Tabela 5.17: Resultados do teste de robustez mono-objetivo para o trecho Praça Raul Soares, com uso do modelo de Akçelik (1991).

$N_{cenarios}$	1	10
Melhor Atraso (s)	16040	16140
Pior Atraso (s)	16040	16140
Média dos Atraso (s)	16040	16140
Mediana dos Atraso (s)	16040	16140
Desvio Padrão	0,00	0,00

[3]Os testes de robustez realizados que utilizaram o modelo de Akçelik (1991) para avaliação dos indivíduos foram feitos na Wokstation HP Z220, enquanto os que utilizaram o modelo de Webster (DENATRAN, 2014) foram executados no PC Lenovo.

Tabela 5.18: Tempos de execução do teste de robustez mono-objetivo o trecho Praça Raul Soares, com uso do modelo de Akçelik (1991).

$N_{cenarios}$	1	10
Número médio de iterações PSO	1000	1000
Tempo médio por iteração PSO (s)	0,153	0,248
Número médio de iterações VNS	5,00	5,00
Tempo médio por iteração VNS (s)	0,151	0,207

As Tabelas 5.19 e 5.20 apresentam as mesmas grandezas para o modelo de Webster (DENATRAN, 2014)no trecho Praça Raul Soares.

Tabela 5.19: Resultados do teste de robustez mono-objetivo para o trecho Praça Raul Soares, com uso do modelo de Webster (DENATRAN, 2014).

$N_{cenarios}$	1	10
Melhor Atraso (s)	105	102
Pior Atraso (s)	125	220
Média dos Atraso (s)	113	130
Mediana dos Atraso (s)	110	126
Desvio Padrão	10,41	27,01

Tabela 5.20: Tempos de execução do teste de robustez mono-objetivo o trecho Praça Raul Soares, com uso do modelo de Webster (DENATRAN, 2014).

$N_{cenarios}$	1	10
Número médio de iterações PSO	1000	1000
Tempo médio por iteração PSO (s)	0,160	0,379
Número médio de iterações VNS	5,33	10,79
Tempo médio por iteração VNS (s)	0,155	0,265

A partir dos resultados obtidos foi possível observar que a melhor solução obtida para 1 cenário tende a não ser a melhor possível ao ser avaliada para 10 cenários. A melhor programação obtida para 1 cenário, com uso do modelo matemático de Akçelik (1991) obteve um atraso de 16040s. Porém, se a mesma programação fosse avaliada para o conjunto de 10 cenários, seu atraso seria de 16042s. O mesmo ocorreu com uso do modelo de Webster (DENATRAN, 2014), onde o atraso obtido para 10 cenários foi de 102s, enquanto a melhor solução obtida para 1 cenário teria o atraso de 103s. Neste caso a diferença foi muito pequena devido ao baixo número de movimentos do trecho.

Trecho Avenida Amazonas I

As Tabelas 5.21 e 5.22 apresentam os resultados e tempos de execução obtidos com o uso do modelo de Akçelik (1991) para o trecho Avenida Amazonas I.

Tabela 5.21: Resultados do teste de robustez mono-objetivo para o trecho Avenida Amazonas I, com uso do modelo de Akçelik (1991).

$N_{cenarios}$	1	10
Melhor Atraso (s)	20602	21103
Pior Atraso (s)	20623	21129
Média dos Atraso (s)	20611	21112
Mediana dos Atraso (s)	20610	21112
Desvio Padrão	5,31	4,62

Tabela 5.22: Tempos de execução do teste de robustez mono-objetivo o trecho Avenida Amazonas I, com uso do modelo de Akçelik (1991).

$N_{cenarios}$	1	10
Número médio de iterações PSO	1000	1000
Tempo médio por iteração PSO (s)	0,216	0,474
Número médio de iterações VNS	9,94	8,39
Tempo médio por iteração VNS (s)	0,246	0,486

As Tabelas 5.23 e 5.24 apresentam os mesmos tipos de resultados, porém obtidos com o modelo de Webster (DENATRAN, 2014).

Tabela 5.23: Resultados do teste de robustez mono-objetivo para o trecho Avenida Amazonas I, com uso do modelo de Webster (DENATRAN, 2014).

$N_{cenarios}$	1	10
Melhor Atraso (s)	516	882
Pior Atraso (s)	704	1329
Média dos Atraso (s)	619	1079
Mediana dos Atraso (s)	638	1084
Desvio Padrão	95,38	103,64

Tabela 5.24: Tempos de execução do teste de robustez mono-objetivo o trecho Avenida Amazonas I, com uso do modelo de Webster (DENATRAN, 2014).

$N_{cenarios}$	1	10
Número médio de iterações PSO	1000	1000
Tempo médio por iteração PSO (s)	0,278	0,782
Número médio de iterações VNS	22,67	18,21
Tempo médio por iteração VNS (s)	0,284	0,636

Os testes apresentados para o trecho Avenida Amazonas I sugerem que o teste de robustez se torna mais importante na medida em que o trecho sob estudo cresce. Enquanto a pior solução obtida com uso do modelo de Akçelik (1991) para 10 cenários apresentou um atraso de 21129s, a melhor solução obtida para 1 cenário, quando avaliada para os 10, obteve um atraso de 21268s (Tabela 5.21). Com o uso do modelo de Webster (DENATRAN, 2014), por sua vez, a melhor solução para 1 cenário apresentou um atraso de 916s ao ser avaliada para 10 cenários, onde a melhor solução obteve 882s (Tabela 5.23).

Vale destacar que o tempo de execução não cresce de forma diretamente proporcional ao número de cenários considerados.

Trecho Avenida Amazonas II

As Tabelas 5.25 e 5.26 apresentam os resultados e tempos de execução, respectivamente, obtidos com o uso do modelo de Akçelik (1991), para o trecho Avenida Amazonas II.

Tabela 5.25: Resultados do teste de robustez mono-objetivo para o trecho Avenida Amazonas II, com uso do modelo de Akçelik (1991).

$N_{cenarios}$	1	10	50
Melhor Atraso (s)	77532	79429	76658
Pior Atraso (s)	77717	79543	76740
Média dos Atraso (s)	77608	79485	76694
Mediana dos Atraso (s)	77613	79478	76697
Desvio Padrão	44,38	25,52	22,73

Tabela 5.26: Tempos de execução do teste de robustez mono-objetivo o trecho Avenida Amazonas II, com uso do modelo de Akçelik (1991).

$N_{cenarios}$	1	10	50
Número médio de iterações PSO	1000	1000	1000
Tempo médio por iteração PSO (s)	0,615	1,346	4,625
Número médio de iterações VNS	111,58	96,94	95,81
Tempo médio por iteração VNS (s)	0,903	1,702	4,916

As Tabelas 5.27 e 5.28 exibem os resultados e tempos de execução, respectivamente, obtidos com o uso do modelo de Webster (DENATRAN, 2014), para o trecho Avenida Amazonas II.

Tabela 5.27: Resultados do teste de robustez mono-objetivo para o trecho Avenida Amazonas II, com uso do modelo de Webster (DENATRAN, 2014).

$N_{cenarios}$	1	10	50
Melhor Atraso (s)	1345	1412	1123
Pior Atraso (s)	1510	3039	1855
Média dos Atraso (s)	1447	1920	1425
Mediana dos Atraso (s)	1487	1827	1337
Desvio Padrão	89,37	340,84	241,89

Tabela 5.28: Tempos de execução do teste de robustez mono-objetivo o trecho Avenida Amazonas II, com uso do modelo de Webster (DENATRAN, 2014).

$N_{cenarios}$	1	10	50
Número médio de iterações PSO	1000	1000	1000
Tempo médio por iteração PSO (s)	0,783	1,617	7,515
Número médio de iterações VNS	306,00	278,12	235,14
Tempo médio por iteração VNS (s)	1,743	2,787	11,941

Os resultados para o trecho Avenida Amazonas II também permitiram observar que para trechos maiores, a falta de robustez pode se tornar um sério problema e, quanto maior o número de cenários de fluxos de veículos utilizados, mais robusta será a programação semafórica obtida. Neste caso, com o uso do modelo de Akçelik (1991), as programações obtidas para 1 e 10 cenários tiveram atrasos de 78460s e 76740s, respectivamente, ao serem avaliadas para 50 cenários, onde a pior solução obtida apresentou um atraso de 76740s (Tabela 5.25). Já com uso do modelo de Webster (DENATRAN, 2014), as programações obtidas para 1 e 10 cenários apresentaram atrasos de 1349s e 1473s ao serem avaliadas para os 50 cenários, enquanto sua melhor solução chegou a um atraso de 1123s (Tabela 5.27).

Avaliação dos Modelo Matemáticos

A partir dos resultados obtidos foi possível observar que as soluções obtidas com o uso do modelo de Akçelik (1991) para a avaliação dos indivíduos tiveram uma boa convergência e que o desvio padrão diminuiu com o aumento do número de cenários utilizados para a avaliação dos indivíduos. Porém, esse comportamento não se repetiu quando o modelo de Webster (DENATRAN, 2014) foi utilizado (vide Tabelas 5.19, 5.23 e 5.27). Dentre outros fatores que podem ter causado a grande variabilidade das soluções, pode-se citar:

- Avaliação de ciclos isolados: Ao contrário do modelo de Akçelik (1991), o modelo de Webster (DENATRAN, 2014) considera os cruzamentos de forma totalmente isolada.

- Uso de cenários com fluxos de veículos nos movimentos saturados: Conforme descrito na Seção 2.4.2, o modelo de Webster (DENATRAN, 2014) depende de condições de tráfego não congestionadas. Contudo, essa situação não foi descartada nos testes realizados, tendo em vista que essa é uma situação possível.

A partir dos resultados obtidos, é sugerido, então, o uso do modelo de Akçelik (1991) para o processo de otimização da programação semafórica. No entanto é necessário que seja feita uma análise mais aprofundada sobre o modelo de Webster (DENATRAN, 2014) para uma conclusão mais assertiva.

5.4.2 Testes Multiobjetivo

O segundo teste de robustez foi feito a partir da variante multiobjetivo do problema de otimização da programação semafórica. Dessa forma, têm-se como objetivos o caso médio (Equação (4.4)) e o pior caso (Equação (4.5)) dos atrasos.

Para este teste, o algoritmo NSGA-II foi executado 11 vezes para cada trecho. As soluções candidatas foram avaliadas para 10 cenários de fluxos de veículos nos movimentos, sendo esses os mesmos utilizados no teste mono-objetivo (Seção 5.4.1). Além disso, foi utilizado apenas o modelo matemático de Akçelik (1991), devido a inconsistência dos resultados apresentados pelo modelo de Webster (DENATRAN, 2014) na Seção 5.4.1.

A Tabela 5.29 apresenta os dados sobre as fronteiras Pareto obtidas.

Tabela 5.29: Resultados das fronteiras Pareto obtidas a partir do teste de robustez multiobjetivo.

Trecho	Pc. Raul Soares	Av. Amazonas I	Av. Amazonas II
Média de pontos	100	75	12
Média de pontos únicos	5	9	12
Maior hipervolume obtido	0,59	0,82	0,93
Menor hipervolume obtido	0,59	0,29	0,21

A Tabela 5.30 exibe o a faixa de valores dos atrasos nas fronteiras Pareto com maiores hipervolumes.

Tabela 5.30: Faixa de valores das soluções nas fronteiras Pareto com maiores hipervolumes obtidas a partir do teste de robustez multiobjetivo.

	Atraso (s) - Caso médio			Atraso (s) - Pior Caso		
Trecho	max	min	% de variação	max	min	% de variação
Praça Raul Soares	16151	16140	0,07	16321	16315	0,04
Avenida Amazonas I	21135	21105	0,14	21732	21713	0,09
Avenida Amazonas II	76829	76735	0,12	79188	79091	0,12

A partir dos resultados foi possível observar a convergência do NSGA-II para fronteiras Pareto contendo poucas soluções únicas. Além disso a variação dos valores de função objetivo das soluções das fronteiras é muito pequenos (vide Tabela 5.30 e detalhes no Apêndice B). Isso sugere a convergência do algoritmo para apenas uma solução. Esse comportamento também justifica a variabilidade dos hipervolumes obtidos (Tabela 5.29) que, por serem normalizados, sofrem grandes perturbações para pequenas alterações na fronteira.

Com isso, é possível concluir que, para os trechos analisados, ao se otimizar o caso médio das avaliações dos indivíduos para os $N_{cenarios}$, otimizou-se simultaneamente o pior caso. Assim, não houve a necessidade de tratar o problema de otimização da programação semafórica como multiobjetivo para esses objetivos. No entanto, vale destacar que outros objetivos podem ser interessantes para o problema.

5.4.3 Considerações Finais

Nas arquiteturas de otimização baseadas no uso de simuladores de tráfego (Figura 4.1) a avaliação para múltiplos cenários de fluxos de veículos nos movimento era inviável devido ao gargalo trazido pelo simulador ao avaliar os indivíduos. A nova arquitetura (Figura 4.2) proposta permitiu essas avaliações, o que garantiu maior robustez e confiança nos resultados obtidos.

5.5 Teste dos Ajustes das Programações Semafóricas

O último experimento realizado neste trabalho está relacionado à segunda arquitetura proposta (Figura 4.3). Essa tem como objetivo ajustar as programações após pequenas variações no fluxo de veículos.

Para isso, o VNS foi executado iterativamente conforme descrito no Algoritmo 15[4]. Assim, foram simulados períodos de tempo ao longo de um dia onde os fluxos de veículos podem variar e de forma a observar como as programações semafóricas se comportam com a mudança dos cenários de fluxos de veículos nos movimentos e como um algoritmo de busca local pode intervir para ajustá-las sempre que necessário.

Algoritmo 15 Teste dos Ajustes das Programações Semafóricas.

1: **function** TESTEARQUITETURA2($solucao_{arquitetura1}, \mathcal{M}, N_{tempos}$)
2: $solucao_{atual} \leftarrow solucao_{arquitetura1}$;
3: **for** $i \leftarrow 1$ até N_{tempos} **do**
4: $cenario_{atual} \leftarrow$ GeradorFluxos($\mathcal{M}$);
5: $avaliacao_{i-1} \leftarrow$ AvaliarSolução($solucao_{atual}, cenario_{atual}$);
6: $solucao_{atual} \leftarrow$ VNS($solucao_{atual}, cenario_{atual}$);
7: **end for**
8: $avaliacao_{N_{tempos}} \leftarrow$ AvaliarSolução($solucao_{atual}, cenario_{atual}$);
9: **end function**

onde:

$solucao_{arquitetura1}$ é a programação obtida ao realizar o processo de otimização pela primeira arquitetura proposta neste trabalho (Figura 4.2);

$\mathcal{M}$ é o conjunto de movimentos existentes no trecho em estudo;

N_{tempos} é o número de iterações do processo, sendo que elas representam os tempos ao longo do dia onde houve variações nos cenários de fluxo de veículos.

5.5.1 Resultados Obtidos

A Tabela 5.31 e a Figura 5.11 apresentam os atrasos obtidos nas avaliações dos indivíduos para o trecho Praça Raul Soares. O tempo 0 indica a solução inicial obtida pelo processo de otimização com uso da primeira arquitetura (Figura 4.2), enquanto os tempos 1 a 10 representam as variações do cenário de fluxo de veículos.

[4]Os ajustes de programações foram executados no PC Lenovo, com uso do modelo matemático de Akçelik (1991) para a avaliação das programações semafóricas candidatas.

Tabela 5.31: Resultados dos ajustes das programações para o trecho Praça Raul Soares.

Tempo	Atraso antes do ajuste (s)	Atraso após o ajuste (s)	% de melhoria
0		16140	
1	16056	16050	0,04
2	16265	16245	0,12
3	16217	16217	0,00
4	16194	16186	0,05
5	16041	16035	0,04
6	16026	16005	0,13
7	16012	16012	0,00
8	15994	15994	0,00
9	15879	15876	0,02
10	16288	16248	0,25

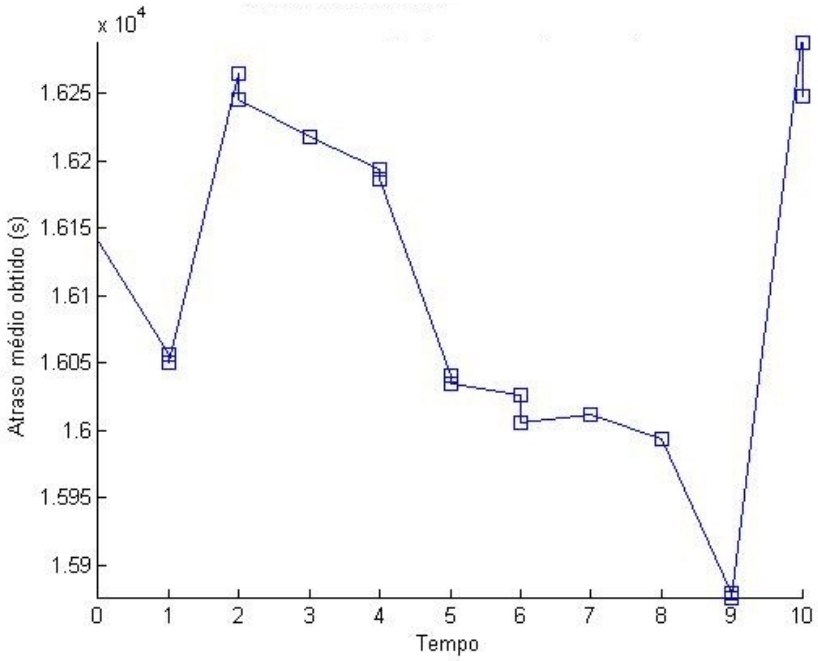


Figura 5.11: Ajuste das programações semafóricas ao longo do tempo para o trecho Praça Raul Soares.

A Tabela 5.32 e a Figura 5.12 descrevem os atrasos obtidos nas avaliações dos indivíduos para o trecho Avenida Amazonas I.

Tabela 5.32: Resultados dos ajustes das programações para o trecho Avenida Amazonas I.

Tempo	Atraso antes do ajuste (s)	Atraso após o ajuste (s)	% de melhoria
0		21103	
1	21324	21315	0,04
2	21250	21059	0,90
3	21115	21090	0,12
4	21117	21065	0,25
5	21490	21442	0,22
6	21155	21081	0,35
7	20958	20825	0,63
8	20899	20831	0,33
9	22114	22036	0,35
10	21400	21208	0,90

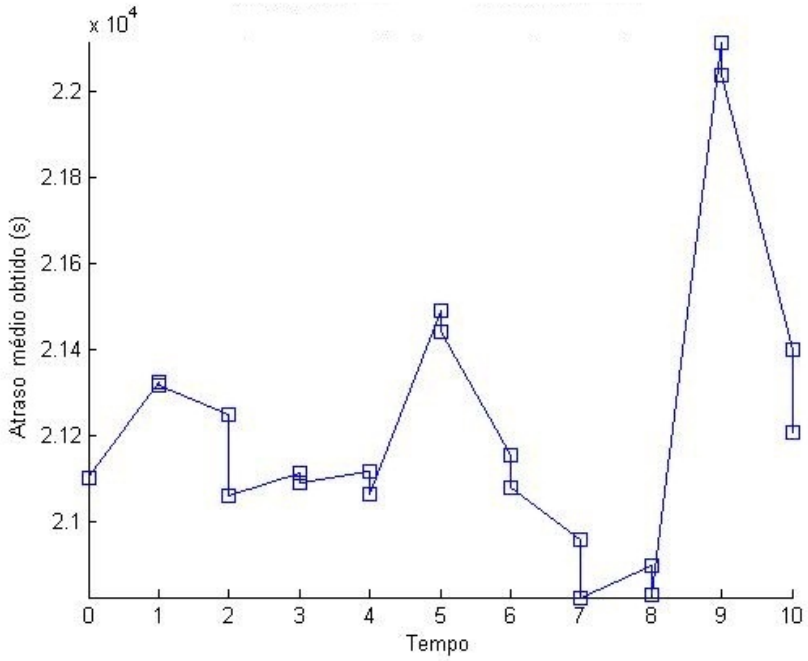


Figura 5.12: Ajuste das programações semafóricas ao longo do tempo para o trecho Avenida Amazonas I.

A Tabela 5.33 e a Figura 5.13 descrevem os atrasos obtidos nas avaliações dos indivíduos para o trecho Avenida Amazonas II.

Tabela 5.33: Resultados dos ajustes das programações para o trecho Avenida Amazonas II.

Tempo	Atraso antes do ajuste (s)	Atraso após o ajuste (s)	% de melhoria
0		77167	
1	75604	75400	0,27
2	75023	74720	0,40
3	78012	77851	0,21
4	76766	76487	0,36
5	76227	76032	0,26
6	78170	77855	0,40
7	78227	77882	0,44
8	77490	77121	0,48
9	77063	76756	0,40
10	77447	77125	0,42

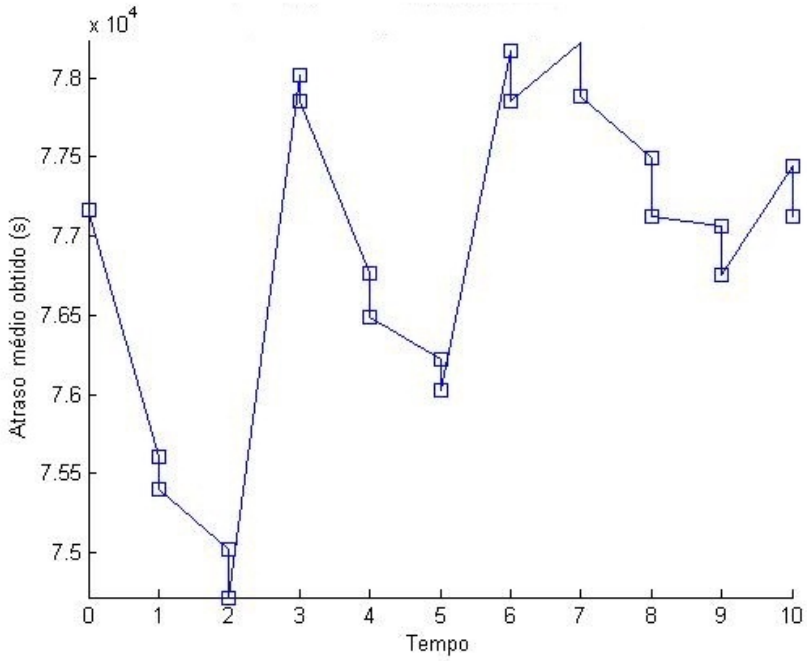


Figura 5.13: Ajuste das programações semafóricas ao longo do tempo para o trecho Avenida Amazonas II.

Por fim, a Tabela 5.34 exibe o tempo de execução do VNS para realizar o ajuste das programações para os três trechos.

Tabela 5.34: Tempos de execução do VNS para os ajustes das programações.

Trecho	Pc. Raul Soares	Av. Amazonas I	Av. Amazonas II
Menor número de iterações	5	8	43
Maior número de iterações	24	70	97
Número médio de iterações	9,20	26,70	67,80
Tempo médio por iteração (s)	0,13	0,24	1,66
Tempo médio total (s)	1,19	6,46	112,21

Os resultados apresentados permitem observar o comportamento das programações semafóricas ao longo de um dia. Em alguns casos, o fluxo de veículos aumenta (ou se torna saturado) e as programações perdem qualidade (aumento do atraso). Em outros casos, os fluxos diminuem, e, consequentemente, os atrasos. Porém, as programações semafóricas foram ajustadas em poucos segundos após notada a alteração de fluxo (vide Tabela 5.34), trazendo diminuições nos atrasos.

5.5.2 Considerações Finais

Ao se considerar que a variação dos fluxos de veículos é pequena em curtos períodos de tempo, é possível apenas realizar um ajuste iterativo nas programações semafóricas para que elas se adequem aos novos cenários. Assim, o ajuste feito pelo algoritmo de busca local precisa de apenas alguns poucos segundos, como visto nos resultados apresentados, e pode diminuir o atraso dos veículos no trecho, mesmo que a mudança do cenário contribua naturalmente para isso. Uma vez que o tempo de ciclo tende a ser consideravelmente superior (neste trabalho foi adotado como $T_{ciclo\_max} = 220s$) e que não ocorrerão mudanças nas programações nesse intervalo de tempo, comprova-se a viabilidade da segunda arquitetura proposta neste trabalho para a otimização das programações semafórica em tempo real.

Capítulo 6

Conclusões

O aumento da frota de veículos trouxe grandes problemas para o gerenciamento do tráfego nas médias e grandes cidades. Nesse contexto, a otimização da programação semafórica é uma das estratégias a serem fortemente consideradas para atenuar os problemas de congestionamento nas redes viárias urbanas, visto seu baixo custo e sua facilidade de implementação.

Dessa forma, foram propostas diversas ferramentas com a finalidade de encontrar boas programações. No entanto, várias delas dependiam de simuladores de tráfego para calcular os atrasos dos veículos e avaliar as soluções candidatas. Como o simulador é lento, devido ao seu alto nível de detalhamento, o processo de otimização podia levar horas. Isso trazia limitações, como, por exemplo, a impossibilidade de se realizar a otimização semafórica em tempo real. Além disso, essa abordagem levava em consideração a avaliação das programações semafóricas candidatas para apenas um cenário de fluxos de veículos nos movimentos, o que trazia dúvidas sobre seu funcionamento no mundo real, onde os fluxos podem variar ao longo do dia.

Este trabalho propôs duas novas arquiteturas para o problema de programação semafórica. A primeira realiza todo o processo de otimização, sendo composta por um algoritmo evolucionário seguido de uma metaheurística de busca local, que tem papel de refinamento. Já a segunda tem como finalidade ajustar as programações semafóricas quando os cenários de fluxos sofrem pequenas variações. Porém, a grande diferença dessas estratégias é que o simulador de tráfego é utilizado apenas no início do procedimento, para geração dos fluxos de veículos que serão utilizados na construção dos modelos matemáticos. Com isso, essas arquiteturas são significativamente mais rápidas que as abordagens anteriores.

Os experimentos realizados vieram a comprovar a importância da avaliação das soluções candidatas para múltiplos cenários de fluxos de veículos (teste de robustez). As programações semafóricas obtidas para vários cenários sempre foram melhores do que as obtidas para um único cenário. Permitir ciclos ligeiramente superiores aos inicialmente estabelecidos (restrição *soft*) também foi uma boa opção. Essa abordagem pode privilegiar alguns ciclos que possuam fluxos de veículos elevados e, dessa forma, diminuir o atraso médio do trecho, sem afetar de forma muito significativa o equilíbrio na rede.

Em relação aos tempos de execução, as duas arquiteturas propostas se mostraram bastante eficientes. Mesmo considerando 10 cenários de fluxos de veículos nos movimentos, a primeira arquitetura conseguiu executar todo o processo de otimização em questão de minutos, sendo cerca de 50 vezes mais rápida que abordagens anteriores. Já a segunda arquitetura foi capaz de corrigir as programações obtidas pela primeira arquitetura para variações dos cenários de fluxos de veículos em apenas alguns segundos, com resultados também satisfatórios.

Com isso, o uso de modelos matemáticos determinísticos remove as limitações relacionadas ao tempo e abre novas perspectivas para se obter programações semafóricas robustas em tempo real.

6.1 Propostas de Continuidade

São sugeridos os seguintes tópicos de estudo para continuidade deste trabalho:

- Testar as programações semafóricas obtidas em simuladores de tráfego, como o SUMO (ITS, 2015) ou AIMSUN (TSS, 2015), a fim de validar a ordinalidade das mesmas.
- Avaliar as limitações do modelo matemático proposto por Akçelik (1991) em relação a precisão dos parâmetros e elaborar uma forma eficiente de calculá-los para os mais diversos tipos de trechos.
- Estudar as razões que levaram a grande variabilidade observada nos resultados obtidos com o modelo de Webster (DENATRAN, 2014).
- Buscar na literatura modelos matemáticos que possibilitem a otimização da programação semafórica levando em conta a sincronização dos semáforos. Caso seja necessário, deve ser proposto um novo modelo capaz de atender à esse objetivo.

- Avaliar as arquiteturas propostas neste trabalho em um ambiente de programação paralela, ou em uma GPU [1], onde os indivíduos ou os cenários de fluxos de veículos nos movimentos possam ser avaliados simultaneamente. Avaliá-las também em sistemas embarcados ou microcontroladores, que geralmente possuem menos recursos computacionais, porém são amplamente utilizados para problemas com natureza de tempo real.
- Buscar novos objetivos a serem tratados no problema multiobjetivo. Dentre outros, a variância dos atrasos nos movimentos, avaliada por Costa (2012), seria importante para garantir que um movimento com maior fluxo de veículos ou com maior velocidade máxima permitida não se torne dominante no processo de otimização em detrimento aos demais.

6.2 Produção Bibliográfica

- Gonzaga, E. W. L., Almeida, P. E. M., and Carrano, E. G. (2015). Otimização da programação semafórica em tempo real com base em modelagem matemática. In *Proc. Congresso Brasileiro de Inteligência Computacional*, Curitiba, Brasil

[1] As Unidades de Processamento Gráfico (do inglês *Graphics Processing Units* ou GPUs) têm incorporado tecnologias para programação de propósito geral e, por sua arquitetura massivamente paralela, possuem uma capacidade de cálculo muito superior a de um processador.

Referências Bibliográficas

Ackley, D. H. (1987). *A Connectionist Machine for Genetic Hillclimbing*. Kluwer Academic Publishers, Norwell, USA.

Akçelik, R. (1981). Traffic signals: capacity and timing analysis. In *Australian Road Research Board (ARRB) Conference*, number 123.

Akçelik, R. (1991). Travel time functions for transport planning purposes: Davidson's function, its time-dependent form and an alternative travel time function. *Australian Road Research*, 21(3):49–59. Revisado em 2000.

Ali, M., Pant, M., and Nagar, A. (2011). Two new approach incorporating centroid based mutation operators for differential evolution. *World Journal of Modelling and Simulation*, 7(1):16–28.

Allsop, R. E. and Charlesworth, J. A. (1977). Traffic in a signal-controlled road network: an example of different signal timings including different routeing. *Traffic Engineering & Control*, 18(Analytic).

Alvarez, I., Poznyak, A., and Malo, A. (2007). Urban traffic control problem via a game theory application. In *Decision and Control, 2007 46th IEEE Conference on*, pages 2957–2961. IEEE.

Azimirad, E., Pariz, N., and Sistani, M. (2010). A novel fuzzy model and control of single intersection at urban traffic network. *IEEE Systems Journal*, 4(1):107–111.

Baker, J. E. (1987). Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application*, pages 14–21, Hillsdale, USA. L. Erlbaum Associates Inc.

Bazan, A. L. C. (1995). A game-theoretic approach to distributed control of traffic signals. In *ICMAS*, volume 12, page 14.

Bazzan, A. L. C. (2005). A distributed approach for coordination of traffic signal agents. *Autonomous Agents and Multi-Agent Systems*, 10(1):131–164.

Ben-Tal, A. and Nemirovski, A. (2000). Robust solutions of linear programming problems contaminated with uncertain data. *Mathematical Programming*, 88(3):411–424.

BHTRANS (2011). Sistema de informações da mobilidade urbana de belo horizonte.

Bingham, E. (2001). Reinforcement learning in neurofuzzy traffic signal control. *European Journal of Operational Research*, 131(2):232–241.

Bonyadi, M. R. and Michalewicz, Z. (2014). A locally convergent rotationally invariant particle swarm optimization algorithm. *Swarm Intelligence*, 8(3):159–198.

Burke, E. K. and Kendall, G. (2014). *Search Methodologies - Introductory Tutorials in Optimization and Decision Support Techniques*. Springer, 2 edition.

Carrano, E. G. (2007). *Algoritmos Evolucionários Eficientes para Otimização de Redes*. PhD thesis, Universidade Federal de Minas Gerais, Belo Horizonte, Brasil.

Cervantes, S. G. S., Piai, J. C., Ramírez, E. F. F., Varasquim, L., and Nagayama, E. (2009). Atefi-um algoritmo para controle semafórico em tempo fixo descentralizado. *Semina: Ciências Exatas e Tecnológicas*, 30(1):41–50.

CET-SP (2010). Portaria 061/2010 do departamento de operação do sistema viário de são paulo.

Cheng, S., Epelman, M., Smith, R. L., et al. (2006). Cosign: A parallel algorithm for coordinated traffic signal control. *IEEE Transactions on Intelligent Transportation Systems*, 7(4):551–564.

Cheung, N. J., Ding, X., and Shen, H. (2014). Optifel: a convergent heterogeneous particle swarm optimization algorithm for takagi–sugeno fuzzy modeling. *IEEE Transactions on Fuzzy Systems*, 22(4):919–933.

Chiu, S. and Chand, S. (1993). Adaptive traffic signal control using fuzzy logic. In *IEEE Transactions on Fuzzy Systems*, pages 1371–1376. IEEE.

Chou, C. and Teng, J. (2002). A fuzzy logic controller for traffic junction signals. *Information Sciences*, 143(1):73–97.

Choy, M. C., Srinivasan, D., and Cheu, R. L. (2003). Cooperative, hybrid agent architecture for real-time traffic signal control. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 33(5):597–607.

Choy, M. C., Srinivasan, D., and Cheu, R. L. (2006). Neural networks for continuous online learning and control. *IEEE Transactions on Neural Networks*, 17(6):1511–1531.

Clerc, M. and Kennedy, J. (2002). The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):58–73.

Coello, C. A. (1999a). An updated suvey of evolutionary multiobjective optimization techniques: State of the art and future trends. pages 3–13, Washington D. C., USA.

Coello, C. A. A. (1999b). A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information Systems*, 1(3):269–308.

CONTRAN (2012). *Manual Brasileiro de Sinalização de Trânsito*. Conselho Nacional de Trânsito, Brasília, Brasil.

Corne, D. W., Knowles, J. D., and Oates, M. J. (2000). The pareto envelope-based selection algorithm for multiobjective optimization. In *Proc. Parallel Problem Solving from Nature Conference*, pages 839–848, Paris, France.

Costa, B. C. (2012). Algoritmos genéticos: estudo, novas técnicas e aplicações. Master's thesis, Centro Federal de Educação Tecnológica de Minas Gerais - CEFET-MG, Belo Horizonte, Brasil.

Czyzźak, P. and Jaszkiewicz, A. (1998). Pareto simulated annealing?a metaheuristic technique for multiple-objective combinatorial optimization. *Journal of Multi-Criteria Decision Analysis*, 7(1):34–47.

D'Acierno, L., Gallo, M., and Montella, B. (2010). An ant colony optimisation (aco) algorithm for solving the local optimisation of signal settings (loss) problem on real-scale networks. Selected proceedings of the 12th World Conference on Transport Research?(Editors: JM Viegas and R. Macario).

Das, S. and Suganthan, P. (2011). Differential evolution: A survey of the state-of-the-art. *IEEE Transactions on Evolutionary Computation*, 15(1):4–31.

Davidson, K. B. (1966). A flow travel time relationship for use in transportation planning. In *Australian Road Research Board (ARRB) Conference*, volume 3, Sydney, Australia.

Davidson, K. B. (1978). The theoretical basis of a flow?travel time relationship for use in transportation planning. In *Australian Road Research Board (ARRB) Conference*, volume 8, pages 32–35, Sydney, Australia.

Davis, T. E. and Principe, J. C. (1991). A simulated annealing like convergence theory for the simple genetic algorithm. In *Proc. International Conference on Genetic Algorithms*, pages 174–181, San Diego, USA. R.K. Belew and L.B. Booker eds.

de Oliveira, D. (2005). Um estudo de coordenação dinâmica de agentes aplicado ao gerenciamento de tráfego veicular urbano. Master's thesis, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brasil.

de Oliveira, D. and Bazzan, A. L. C. (2006). Traffic lights control with adaptive group formation based on swarm intelligence. In *Ant Colony Optimization and Swarm Intelligence*, pages 520–521. Springer.

Deb, K. (2000). An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186(2-4):311–338.

Deb, K. (2008). *Introduction to Evolutionary Multiobjective Optimization*, volume 5252 of *Lecture Notes in Computer Science*. Springer.

Deb, K., Pratap, A., Agrawal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6:182–197.

DENATRAN (1984). *Manual de Semáforos*. Departamento Nacional de Trânsito, Brasília, Brasil, 2a edition.

DENATRAN (2014). *Manual Brasileiro de Sinalização de Trânsito Volume V - Sinalização Semafórica*. Departamento Nacional de Trânsito, Brasília, Brasil.

dos Santos, C. K., Espíndola, R. P., and Evsukoff, A. G. (2015). Development of recurrent fuzzy systems with external feedback. *Proc. IEEE Latin America Transactions*, 13(1).

Downie, A. (2008). The world's worst traffic jams. Disponível em `http://content.time.com/time/world/article/0,8599,1733872,00.html`. Acessado em Dezembro de 2015.

Eshelman, L. J., Caruana, R. A., and Schaffer, J. D. (1989). Biases in the crossover landscape. In *Proc. International Conference on Genetic Algorithms*, pages 10–19, San Francisco, USA. Morgan Kaufmann Publishers Inc.

Evers, G. (2009). An automatic regrouping mechanism to deal with stagnation in particle swarm optimization. Master's thesis, The University of Texas - Pan American, Department of Electrical Engineering, Edinburg, USA.

Feo, T. A. and Resende, M. G. C. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67–71.

Fonseca, C. M. (1995). *Multiobjective genetic algorithms with applications to control engineering problems*. PhD thesis, University of Sheffield, Sheffield, UK.

Fonseca, C. M. and Fleming, P. J. (1993). Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In *Proc. International Conference on Genetic Algorithms*, San Mateo, USA.

Fonseca, C. M., Knowles, J. D., Thiele, L., and Zitzler, E. (2005). A tutorial on the performance assessment of stochastic multiobjective optimizers. In *Third International Conference on Evolutionary Multi-Criterion Optimization (EMO 2005)*, volume 216, page 240.

Foy, M. D., Benekohal, R. F., and Goldberg, D. E. (1992). Signal timing determination using genetic algorithms. *Transportation Research Record*, (1365).

Gendreau, M. and Potvin, J. (2010). *Handbook of Metaheuristics*, volume 146. Springer, 2 edition.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13(5):533–549.

Glover, F. and Kochenberger, G. A. (2003). *Handbook of Metaheuristics*. Kluwer Academic Publishers.

Gokulan, B. P. and Srinivasan, D. (2010). Distributed geometric fuzzy multiagent urban traffic signal control. *IEEE Transactions on Intelligent Transportation Systems*, 11(3):714–727.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co. Inc., Boston, USA, 1st edition.

Golding, S. (1977). On davidson's flow/travel time relationship. *Australian Road Research Board (ARRB) Conference*, 7(3):36–37.

Gonzaga, E. W. L., Almeida, P. E. M., and Carrano, E. G. (2015). Otimização da programação semafórica em tempo real com base em modelagem matemática. In *Proc. Congresso Brasileiro de Inteligência Computacional*, Curitiba, Brasil.

Guberinic, S., Senborn, G., and Lazic, B. (2007). *Optimal traffic control: urban intersections*. CRC Press.

Guojiang, S. and Youxian, S. (2005). Fuzzy neural network control technique and its application in a complex intersection. In *Neural Networks and Brain, 2005. ICNN&B'05. International Conference on*, volume 2, pages 970–974. IEEE.

Hadi, M. A. and Wallace, C. E. (1993). Hybrid genetic algorithm to optimize signal phasing and timing. *Transportation Research Record*, (1421):104?112.

Haight, F. A. (1967). *Handbook of the Poisson Distribution*. John Wiley & Sons, 1 edition.

Hansen, M. P. e Jaszkiewicz, A. (1998). Evaluating the quality of approximations to the non-dominated set.

He, J. and Hou, Z. (2012). Ant colony algorithm for traffic signal timing optimization. *Advances in Engineering Software*, 43(1):14–18.

Henry, J. J., Farges, J. L., and Gallego, J. L. (1998). Neuro-fuzzy techniques for traffic control. *Control Engineering Practice*, 6(6):755–761.

Hoare, C. A. R. (1961). Algorithm 64: Quicksort. *Commun. ACM*, 4(7):321–.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. The University of Michigan Press.

Hong, Y. S., JongSoo, K., Son, J., and Park, C. (1999). Estimation of optimal green time simulation using fuzzy neural network. In *IEEE Transactions on Fuzzy Systems*, volume 2, pages 761–766. IEEE.

Horn, J., Nafpliotis, N., and Goldberg, D. E. (1994). A niched pareto genetic algorithm for multiobjective optimization. In *Proc. IEEE International Congress on Evolutionary Computation*, Orlando, USA.

IBGE (2000). Censo demográfico 2000. Disponível em `www.ibge.gov.br/home/estatistica/populacao/censo2000/tabelabrasil111.shtm`.

IBGE (2010). Censo demográfico 2010. Disponível em `http://www.ibge.gov.br/home/estatistica/populacao/censo2010/default.shtm`.

INCT (2013). Evolução da frota de automóveis e motos no brasil 2001 - 2012. Disponível em `www.observatoriodasmetropoles.net/download/auto_motos2013.pdf`. Acessado em Dezembro de 2015.

ITS (1993). Dracula - dynamic route assignment combining user learning and microsimulation. Disponível em `www.its.leeds.ac.uk/software/dracula/`. Acessado em Dezembro de 2015.

ITS (2015). Sumo - simulation of urban mobility. Disponível em `www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931_read-41000/`. Acessado em Dezembro de 2015.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Proc. IEEE International Conference on Neural Networks IV*, volume 4, pages 1942–1948, Perth, Austrália.

Kesur, K. B. (2009). Advances in genetic algorithm optimization of traffic signals. *Journal of Transportation Engineering*.

Kim, J. (1997). A fuzzy logic control simulator for adaptive traffic management. In *IEEE Transactions on Fuzzy Systems*, volume 3, pages 1519–1524, Barcelona, Spain. IEEE.

Kirkpatrick, S., Gelatt Jr, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.

Knowles, J. and Corne, D. (1999). The pareto archived evoution strategy: a new baseline for pareto multiobjective optimisation. In *Proc. IEEE International Congress on Evolutionary Computation*, pages 98–105, Washington, USA.

Knowles, J. and Corne, D. (2002). On metrics for comparing nondominated sets. In *Proc. IEEE International Congress on Evolutionary Computation*, volume 1, pages 711–716.

Kosonen, I. (2003). Multi-agent fuzzy signal control based on real-time simulation. *Transportation Research Part C: Emerging Technologies*, 11(5):389–403.

Lämmer, S. and Helbing, D. (2008). Self-control of traffic lights and vehicle flows in urban road networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(04):1–30.

Lee, J. and Lee-Kwang, H. (1999). Distributed and cooperative fuzzy controllers for traffic intersections group. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 29(2):263–271.

Li, H. and Zhang, Q. (2009). Multiobjective optimization problems with complicated pareto sets, moea/d and nsga-ii. *IEEE Transactions on Evolutionary Computation*, 13(2):284–302.

Lovbjerg, M. and Krink, T. (2002). Extending particle swarm optimisers with self-organized criticality. In *wcci*, pages 1588–1593. IEEE.

López, S., Hernández, P., Hernández, A., and García, M. (1999). Artificial neural networks as useful tools for the optimization of the relative offset between two consecutive sets of traffic lights. In *Engineering Applications of Bio-Inspired Artificial Neural Networks*, pages 795–804. Springer.

Ma, W., Geng, D., and Yan, Y. (2012). Multi-phase fuzzy control of single intersection in traffic system based on genetic algorithm. *International Journal of Innovative Computing Information and Control*, 8(5):3387–3397.

Mallipeddia, R., Suganthana, P. N., Panb, Q. K., and Tasgetirenc, M. F. (2011). *Differential evolution algorithm with ensemble of parameters and mutation strategies*, volume 11. Elsevier.

Masterton, T. and Topiwala, D. (2008). Multi-agent traffic light optimisation and coordination. *Thales Research and Technology: White Papers*, 2.

Miller, B. L. and Goldberg, D. E. (1995). Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, 9(3):193–212.

Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers and Operations Research*, 24(11):1097–1100.

Murat, Y. S. and Gedizlioglu, E. (2005). A fuzzy logic multi-phased signal control model for isolated junctions. *Transportation Research Part C: Emerging Technologies*, 13(1):19–36.

Neto, T. S. (2009). Desenvolvimento de um sig de código-aberto para simulação microscópica de tráfego urbano. Master's thesis, Centro Federal de Educação Tecnológica de Minas Gerais - CEFET-MG, Belo Horizonte, Brasil.

Nobile, M. S., Besozzi, D., Cazzaniga, P., Mauri, G., and Pescini, D. (2012). A gpu-based multi-swarm pso method for parameter estimation in stochastic biological systems exploiting discrete-time target series. In *Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*, pages 74–85. Springer.

Oda, T., Otokita, T., Tsugui, T., Kohno, M., and Mashiyama, Y. (1996). Optimization of signal control parameters using a genetic algorithm. In *Intelligent Transportation: Realizing the Future. Abstracts of the Third World Congress on Intelligent Transport Systems.*

Oliveira, F. B. (2009). Geração de tempos semafóricos utilizando inteligência computacional em sig convencionais. Master's thesis, Centro Federal de Educação Tecnológica de Minas Gerais - CEFET-MG, Belo Horizonte, Brasil.

OpenJUMP (2014). Openjump. Disponível em `www.openjump.org/index.html`. Acessado em Dezembro de 2015.

Park, B. (1998). *Development of genetic algorithm-based signal optimization program for oversaturated intersections.* PhD thesis, Texas A&M University, Galveston, USA.

Pearl, J. (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley Longman Publishing Co. Inc., Boston, USA.

Pereira, G. and Ribeiro, M. V. M. (2007). Controle de tráfego em tempo real: Novos paradigmas, dificuldades e primeiros resultados ? o caso do controle inteligente de tráfego (cit). In *Proc. Congresso Brasileiro de Transporte e Trânsito*, Maceió, Brasil.

Pillai, R. S., Rathi, A. K., and Cohen, S. L. (1998). A restricted branch-and-bound approach for generating maximum bandwidth signal timing plans for traffic networks. *Transportation Research Part B: Methodological*, 32(8):517–529.

PMV (2015). Semáforos: Sincronia e programação em tempo real. Disponível em `www.vitoria.es.gov.br/cidadao/semaforos-sincronia-e-programacao-em-tempo-real`.

Porche, I. and Lafortune, S. (1998). A game-theoretic approach to signal coordination.

Price, K., Storn, R. M., and Lampinen, J. A. (2006). *Differential evolution: a practical approach to global optimization.* Springer Science and Business Media.

Putha, R., Quadrifoglio, L., and Zechman, E. (2012). Comparing ant colony optimization and genetic algorithm approaches for solving traffic signal coordination under oversaturation conditions. *Computer-Aided Civil and Infrastructure Engineering*, 27(1):14–28.

Radcliffe, N. (1990). *Genetic Neural Networks on MIMD Computers*. PhD thesis, University of Edinburgh, Edinburgh, UK.

Ramos, R. M., Saldanha, R. R., Takahashi, R. H. C., and Moreira, F. J. S. (2003). The real-biased multiobjective genetic algorithm and its application to the design of wire antennas. *IEEE Transactions on Magnetics*, 39:1329–1332.

Renfrew, D. and Yu, X. (2009). Traffic signal control with swarm intelligence. In *Natural Computation, 2009. ICNC'09. Fifth International Conference on*, volume 3, pages 79–83. IEEE.

Robertson, D. I. (1969). Transyt - a traffic network study tool. Disponível em `https://trlsoftware.co.uk/products/junction_signal_design/transyt`. Acessado em Dezembro de 2015.

Roozemond, D. A. (2001). Using intelligent agents for pro-active, real-time urban intersection control. *European Journal of Operational Research*, 131(2):293–301.

Rose, G., Taylor, M. A. P., and Tisato, P. (1989). Estimating travel time functions for urban roads: options and issues. *Transportation Planning and Technology*, 14(1):63–82.

Rouphail, N. M., Park, B. B., and Sacks, J. (2000). Direct signal timing optimization: Strategy development and results. In *In XI Pan American Conference in Traffic and Transportation Engineering*. Citeseer.

Saito, M. and Fan, J. (2000). Artificial neural network–based heuristic optimal traffic signal timing. *Computer-Aided Civil and Infrastructure Engineering*, 15(4):293–307.

Saka, A. A., Anandalingam, G., and Garber, N. J. (1986). Traffic signal timing at isolated intersections using simulation optimization. In *Proceedings of the 18th conference on Winter simulation*, pages 795–801. ACM.

Shamshirband, S. S., Shirgahi, H., Gholami, M., and Kia, B. (2008). Coordination between traffic signals based on cooperative. *World Applied Sciences Journal*, 5(5):525–530.

Sheffi, Y. and Powell, W. B. (1983). Optimal signal settings over transportation networks. *Journal of Transportation Engineering*, 109(6):824–839.

Shen, G. and Kong, X. (2009). Study on road network traffic coordination control technique with bus priority. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 39(3):343–351.

Soares, G. L. (1997). Algoritmos genéticos: estudo, novas técnicas e aplicações. Master's thesis, Universidade Federal de Minas Gerais, Belo Horizonte, Brasil.

Spall, J. C. and Chin, D. C. (1997). Traffic-responsive signal timing for system-wide traffic control. *Transportation Research Part C: Emerging Technologies*, 5(3):153–163.

Srinivas, N. and Deb, K. (1994). Muiltiobjective optimization using nondominated sorting in genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 2(3):221–248.

Srinivasan, D. and Choy, M. C. (2006). Cooperative multi-agent system for coordinated traffic signal control. In *IEEE Transactions on Intelligent Transportation Systems*, volume 153, pages 41–50. IET.

Srinivasan, D., Choy, M. C., and Cheu, R. L. (2006). Neural networks for real-time traffic signal control. *IEEE Transactions on Intelligent Transportation Systems*, 7(3):261–272.

Storn, R. and Price, K. (1997). Differential evolution: A simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359.

Syswerda, G. (1989). Uniform crossover in genetic algorithms. In *Proc. International Conference on Genetic Algorithms*, pages 2–9, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Sánchez, J., Galán, M., and Rubio, E. (2008). Applying a traffic lights evolutionary optimization technique to a real case:?las ramblas? area in santa cruz de tenerife. *Proc. IEEE International Congress on Evolutionary Computation*, 12(1):25–40.

Takahashi, R. H. C. (2007). Otimização escalar e vetorial. Disponível em `www.mat.ufmg.br/~taka/`. Acessado em Dezembro de 2015.

Tanomaru, J. (1995). Motivação, fundamentos e aplicações de algoritmos genéticos. In *Proc. Congresso Brasileiro de Redes Neurais*, Curitiba, Brazil.

Tisato, P. (1991). Suggestions for an improved davidson travel time function. *Australian Road Research Board (ARRB) Conference*, 21(2).

Trabia, M. B., Kaseko, M. S., and Ande, M. (1999). A two-stage fuzzy logic controller for traffic signals. *Transportation Research Part C: Emerging Technologies*, 7(6):353–367.

TRB (1965). *Highway Capacity Manual 1965*. Highway Research Board of the Division of Engineering and Industrial Research, National Academy of Sciences-National Research Council, USA.

TRB (2000). *Highway Capacity Manual 2000*. Transportation Research Board, USA.

TRB (2010). *National Cooperative Highway Research Program: Traffic Signal Retiming Practices in the United States*. Transportation Research Board, USA.

Trelea, I. C. (2003). The particle swarm optimization algorithm: convergence analysis and parameter selection. *Information processing letters*, 85(6):317–325.

TSS (2015). Aimsun. Disponível em `www.aimsun.com/wp/`. Acessado em Dezembro de 2015.

Turky, A. M., Ahmad, M. S., and Yusoff, M. Z. M. (2009). The use of genetic algorithm for traffic light and pedestrian crossing control. *International Journal of Computer Science and Network Security*, 9(2):88–96.

van den Bergh, F. (2001). *An Analysis of Particle Swarm Optimizers*. PhD thesis, University of Pretoria, Faculty of Natural and Agricultural Science, Pretoria, South Africa.

van den Bergh, F. and Engelbrecht, A. P. (2010). A convergence proof for the particle swarm optimiser. *Fundamenta Informaticae*, 105(4):341–374.

Vilanova, L. (2005). Programação de um semáforo usando o método do grau de saturação. Disponível em `www.sinaldetransito.com.br/artigos/saturacao.pdf`. Acessado em Dezembro de 2015.

Vogel, A., Goerick, C., and Von Seelen, W. (2000). Evolutionary algorithms for optimizing traffic signal operation. In *Proceedings of the European symposium on intelligent techniques (ESIT)*, pages 83–91. Citeseer.

von Zuben, F. J. (2009). Computação evolutiva: Uma abordagem pragmática. Disponível em `ftp.dca.fee.unicamp.br/pub/docs/vonzuben/tutorial/tutorialEC.pdf`. Acessado em Dezembro de 2015.

Webster, F. V. (1958). Traffic signal settings. (39).

Wright, A. H. (1991). Genetic algorithms for real parameter optimization. In *Foundations of Genetic Algorithms*, pages 205–218. Morgan Kaufmann Publishers Inc.

Xie, X., Barlow, G. J., Smith, S. F., and Rubinstein, Z. B. (2011). Selfscheduling agents for real-time traffic signal control. Technical report, Technical Report TR-RI-11-06, Robotics Institute, Carnegie Mellon University.

Xie, X., Zhang, W., and Yang, Z. (2005). A dissipative particle swarm optimization. *Congress on Evolutionary Computation*, pages 1456–1461.

Xinchao, Z. (2010). A perturbed particle swarm algorithm for numerical optimization. *Applied Soft Computing*, 10(1):119–124.

Yan, F., Dridi, M., and El Moudni, A. (2008). Control of traffic lights in intersection: A new branch and bound approach. In *Service Systems and Service Management, 2008 International Conference on*, pages 1–6. IEEE.

Yuan, Y., Xu, H., and Wang, B. (2014). An improved nsga-iii procedure for evolutionary many-objective optimization. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO '14, pages 661–668, New York, USA. ACM.

Zarandi, M. H. F. and Rezapour, S. (2009). A fuzzy signal controller for isolated intersections. *Journal of Uncertain Systems*, 3(3):174–182.

Zechman, E., Quadrifoglio, L., and Putha, R. (2010). Ant colony optimization algorithm for signal coordination of oversaturated traffic networks. Technical report, Southwest Region University Transportation Center, Texas Transportation Institute.

Zhan, Z., Zhang, J., Li, Y., and Chung, H. S. (2009). Adaptive particle swarm optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 39(6):1362–1381.

Zhang, Q. and Li, H. (2007). Moea/d: A multi-objective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731.

Zitzler, E., Laumanns, M., and Thiele, L. (2001). Spea2: Improving the strength pareto evolutionary algorithm. Disponível em `www.tik.ee.ethz.ch/sop/publicationListFiles/zlt2001a.pdf`. Acessado em Dezembro de 2015.

Zitzler, E. and Thiele, L. (1998). *Multiobjective optimization using evolutionary algorithms - A comparative case study*, volume 1498 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg.

Zitzler, E. and Thiele, L. (1999). Multiobjective evolutionary algorithms: a comparative case study and the Strength Pareto approach. *IEEE Transactions on Evolutionary Computation*, 3:257–271.

Apêndice A

Melhores Programações Semafóricas Obtidas nos Testes de Penalidades

Tabela A.1: Melhores programações semafórica obtidas para o trecho Praça Raul Soares.

(a) $T_{penalidade} = 0s$

ciclo	semáforo	t_{verde} (s)
1	1	100
1	2	77
1	3	92
2	4	100
2	5	93
3	6	75
3	7	100
3	8	98
4	9	100
4	10	100
5	11	100
5	12	88

(b) $T_{penalidade} = 10s$

ciclo	semáforo	t_{verde} (s)
1	1	81
1	2	60
1	3	70
2	4	100
2	5	93
3	6	57
3	7	79
3	8	76
4	9	100
4	10	100
5	11	100
5	12	88

Tabela A.2: Melhores programações semafórica obtidas para o trecho Avenida Amazonas I.

(a) $T_{penalidade} = 0s$

ciclo	semáforo	t_{verde} (s)
1	1	100
1	2	46
1	3	82
2	4	100
2	5	47
2	6	91
3	7	100
3	8	58
3	9	99
3	10	54
4	11	100
4	12	57
4	13	100
5	14	80
5	15	100
6	16	81
6	17	77
6	18	100
6	19	78

(b) $T_{penalidade} = 10s$

ciclo	semáforo	t_{verde} (s)
1	1	94
1	2	43
1	3	74
2	4	90
2	5	42
2	6	79
3	7	69
3	8	38
3	9	65
3	10	36
4	11	83
4	12	47
4	13	81
5	14	80
5	15	100
6	16	51
6	17	46
6	18	64
6	19	48

Tabela A.3: Melhores programações semafórica obtidas para o trecho Avenida Amazonas II.

(a) $T_{penalidade} = 0s$

ciclo	semáforo	t_{verde} (s)
1	1	100
1	2	44
1	3	75
2	4	100
2	5	45
2	6	84
3	7	100
3	8	54
3	9	92
3	10	50
4	11	100
4	12	55
4	13	96
5	14	87
5	15	100
6	16	93
6	17	78
6	18	100
6	19	79
7	20	100
7	21	64
7	22	89
8	23	87
8	24	100
9	25	79
9	26	100
9	27	72
10	28	100
10	29	75
11	30	100
11	31	94
12	32	100
12	33	76
12	34	93
12	35	89
13	36	100
13	37	85
13	38	41
14	39	100
14	40	99
14	41	75
15	42	92
15	43	100
15	44	68
15	45	54
16	46	88
16	47	100
16	48	44
16	49	58
17	50	89
17	51	100
17	52	42
18	53	89
18	54	100
18	55	53
18	56	53

(b) $T_{penalidade} = 10s$

ciclo	semáforo	t_{verde} (s)
1	1	99
1	2	42
1	3	71
2	4	94
2	5	41
2	6	76
3	7	72
3	8	38
3	9	63
3	10	35
4	11	86
4	12	46
4	13	79
5	14	86
5	15	100
6	16	58
6	17	50
6	18	67
6	19	50
7	20	86
7	21	52
7	22	73
8	23	86
8	24	100
9	25	66
9	26	86
9	27	60
10	28	100
10	29	75
11	30	100
11	31	94
12	32	65
12	33	48
12	34	57
12	35	55
13	36	96
13	37	77
13	38	38
14	39	81
14	40	75
14	41	55
15	42	67
15	43	78
15	44	49
15	45	40
16	46	62
16	47	75
16	48	31
16	49	41
17	50	81
17	51	93
17	52	37
18	53	62
18	54	75
18	55	37
18	56	38

(c) $T_{penalidade} = 1000s$

ciclo	semáforo	t_{verde} (s)
1	1	100
1	2	41
1	3	71
2	4	94
2	5	41
2	6	76
3	7	75
3	8	38
3	9	62
3	10	33
4	11	86
4	12	46
4	13	79
5	14	86
5	15	100
6	16	54
6	17	47
6	18	60
6	19	47
7	20	87
7	21	53
7	22	71
8	23	87
8	24	100
9	25	66
9	26	88
9	27	57
10	28	100
10	29	75
11	30	100
11	31	95
12	32	61
12	33	45
12	34	51
12	35	51
13	36	96
13	37	77
13	38	38
14	39	80
14	40	75
14	41	56
15	42	59
15	43	70
15	44	44
15	45	35
16	46	61
16	47	74
16	48	32
16	49	41
17	50	81
17	51	93
17	52	37
18	53	62
18	54	73
18	55	36
18	56	37

Apêndice B

Fronteiras Pareto Obtidas nos Testes de Robustez Multiobjetivos

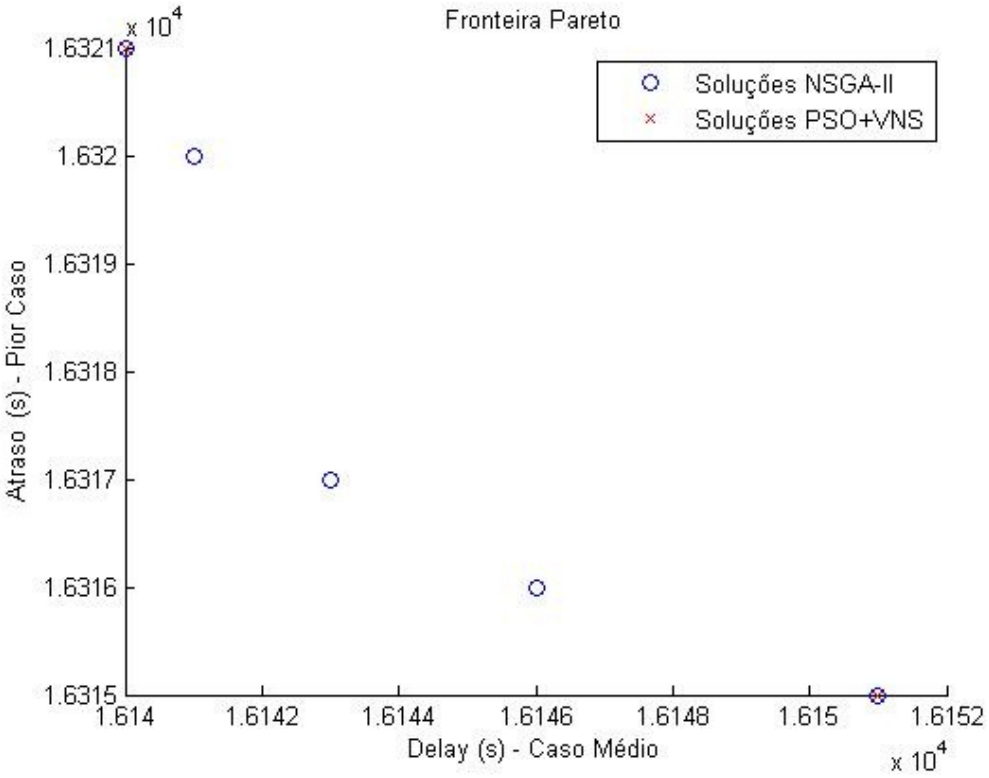
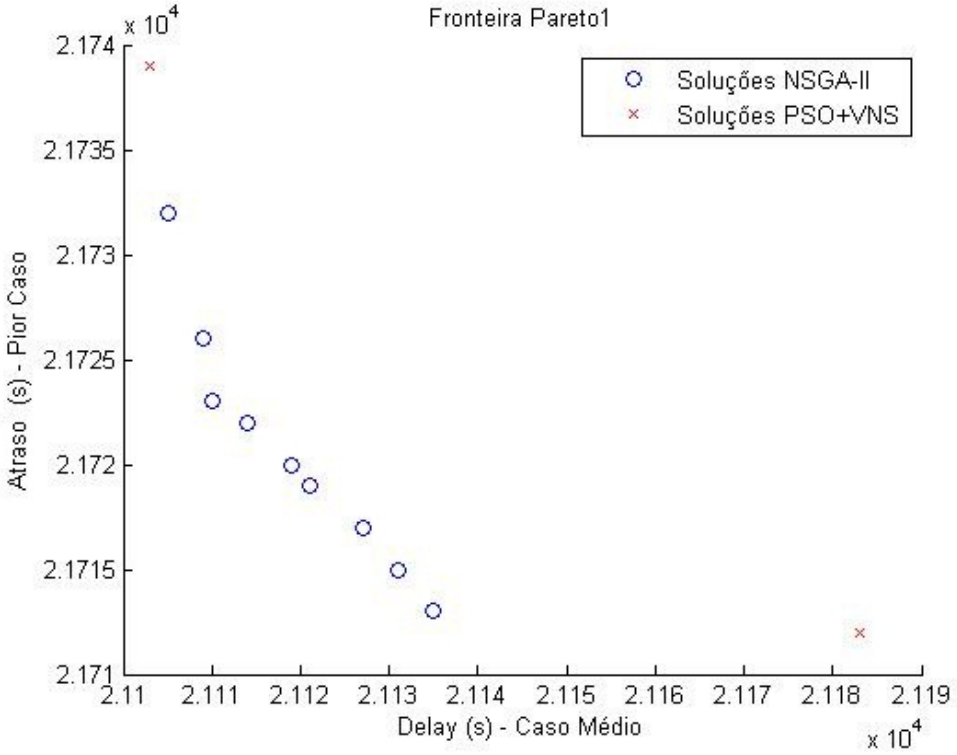
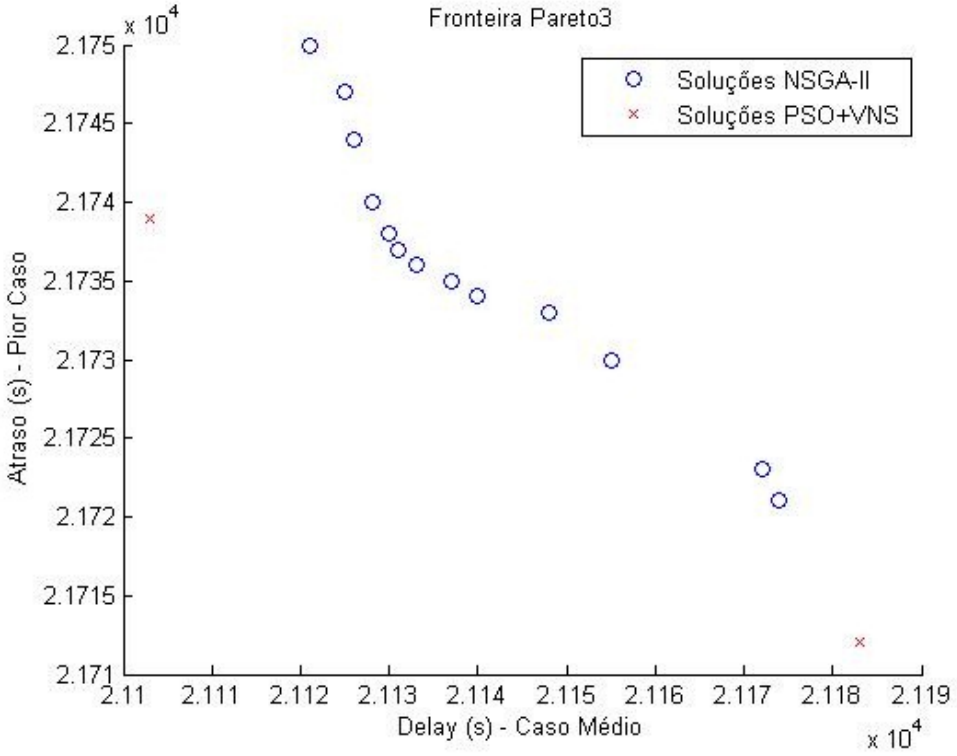


Figura B.1: Fronteira Pareto obtida para o Trecho Praça Raul Soares.

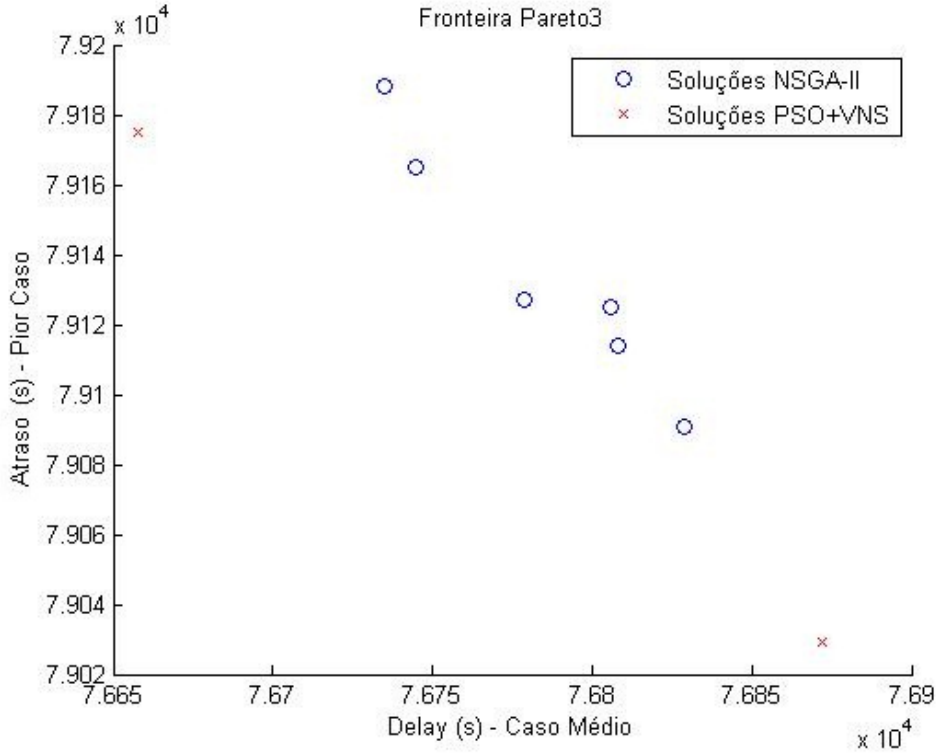


(a) Fronteira com maior hipervolume.

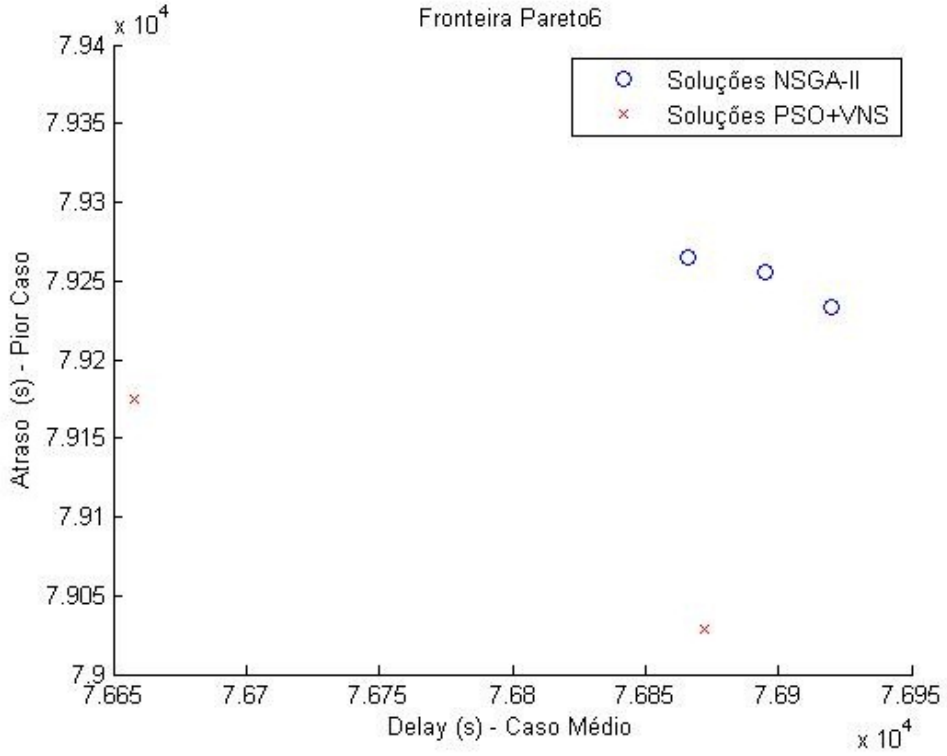


(b) Fronteira com menor hipervolume.

Figura B.2: Fronteiras Pareto obtidas para o Trecho Avenida Amazonas I.



(a) Fronteira com maior hipervolume.



(b) Fronteira com menor hipervolume.

Figura B.3: Fronteiras Pareto obtidas para o Trecho Avenida Amazonas II.