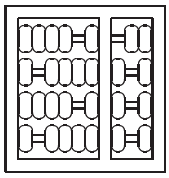


INSTITUTO DE COMPUTAÇÃO
UNIVERSIDADE ESTADUAL DE CAMPINAS

Um Algoritmo Genético para Projeto de Rede de Telecomunicações

S. Livramento *A. V. Moura*
F. K. Miyazawa *M. M. Harada*
R. A. Miranda

Technical Report - IC-03-024 - Relatório Técnico

November - 2003 - Novembro

Um Algoritmo Genético para Projeto de Rede de Telecomunicações *

Silvana Livramento † Arnaldo Vieira Moura † Flávio Keidi Miyazawa †
Mário Massato Harada‡ Rogério Albertoni Miranda‡

Resumo

Este relatório descreve um Algoritmo Genético (AG) desenvolvido para resolver um problema de projeto de rede telefônica. O problema consiste em particionar uma grande área de projeto urbana em pequenas seções de serviço, as quais podem ser controladas por um único dispositivo de telecomunicação padrão. O AG incorpora informações geométricas e topológicas da área de projeto operando diretamente com uma matriz de pontos de demanda geográficos. Os resultados computacionais mostraram que esta é uma técnica promissora para particionar a área de projeto em seções de serviço e para posicionar o dispositivo de controle dentro de cada seção. Os testes foram realizados com instâncias reais tomadas de grandes áreas da cidade de São Paulo.

Palavras chaves: projeto de rede, algoritmo genético, telecomunicações.

1 Introdução

Neste capítulo detalhamos o trabalho que está sendo desenvolvido com o algoritmo genético (AG), para o problema de "Definições das Seções de Serviço e Posicionamento de Armários". Este problema é dado por um conjunto S de triplas $(x, y, d) \in \mathbb{R}^3$, onde x e y são as coordenadas de localização de um ponto de demanda, e d representa a demanda, em termos de linhas de comunicação, para ser atendida nesta localização. Uma solução é uma partição π de S, cada membro de π é chamado de seção de serviço. Esta partição π tem que satisfazer algumas restrições. Dentro do casco convexo de cada seção de serviço, temos que posicionar um armário de distribuição. Inicialmente, não sabemos quantas seções de serviço existem.

O custo de uma seção de serviço é medido pelo custo do armário de distribuição, mais uma estimativa do custo dos cabos de comunicação ligando o armário de distribuição à cada ponto de demanda. Assim, o custo de uma seção de serviço s é definido por $\phi(s) =$

[1]Pesquisa parcialmente financiada pelo CNPq (Proc. 300301/98-7, 470608/01–3, 478818/03–3) e CPqD.

[2]Instituto de Computação — Universidade Estadual de Campinas, Caixa Postal 6176 — 13084–971 — Campinas–SP — Brasil, {silvana.livramento,arnaldo,fkm}@ic.unicamp.br.

[3]Centro de Pesquisa e Desenvolvimento em Telecomunicações - CPqD, Rodovia Campinas - Mogi-Mirim, km 118,5 — 13086–902 — Campinas–SP — Brasil,{harada,rogeriom}@cpqd.com.br.

$c_a + f \sum_{i \in s}[d_i \times dist(p_a, p_i)]$, onde c_a é o custo fixo de um armário de distribuição, p_i e d_i são a posição e a demanda do ponto i, respectivamente, p_a é a localização do armário de distribuição na seção de serviço s, $dist$ é a distância euclideana no plano, e $f = \sigma\beta$ é uma constante, onde σ representa o custo médio por unidade de cabo e $\beta > 1$ é um fator de correção que é aplicado na distância, pois na realidade o cabeamento não segue o caminho mais curto. As constantes σ e β foram obtidas fazendo a média sobre a rede existente em áreas de projeto similares. Assim, o custo de uma solução π é o valor $w(\pi) = \sum_{s \in \pi} \phi(s)$. O objetivo é achar uma solução π que minimize $w(\pi)$. Além disto, a demanda total dentro de uma seção de serviço está sujeita a valores mínimo e máximo. Cada seção de serviço $s \in \pi$ deve satisfazer $K_a \times C \leq D_s \leq K_A \times C$. Através deste relatório, usaremos C como a capacidade de um armário de distribuição, K_A como o fator de carga máxima, K_a como o fator de carga mínima e D_s como a demanda total em s (i.e. $D_s = \sum_{i \in s} d_i$).

Na seção 2 explicamos em linhas gerais a idéia que norteia o funcionamento de um algoritmo genético, quais são seus principais passos, operações e características.

As idéias e os detalhes de modelagem das operações do algoritmo genético foram listados na seção 3. Nesta seção tratamos a vizinhança entre pontos, ou seja, como adquirir informações geométricas e geográficas de um conjunto de coordenadas, para que o algoritmo gere soluções com uma geometria razoável. Apresentamos, também, a forma como as soluções são modeladas como um indivíduo do AG, o tratamento das restrições de capacidade dos armários de distribuição, a função de aptidão, e por fim os operadores de seleção, mutação e cruzamento.

Os detalhes de implementação destas idéias estão apresentados na seção 4. Nesta seção descrevemos o propósito das principais classes utilizadas, juntamente com alguns atributos e métodos mais relevantes. Por fim, na seção 5 listamos uma série de testes de parâmetros para uma determinada instância do problema e também ilustramos as soluções obtidas para os dados da Vila Mariana e do Bairro da Liberdade.

2 Funcionamento

Algoritmos genéticos (AG) são métodos heurísticos cujo princípio de funcionamento tem uma analogia direta com o processo evolutivo natural, baseados na teoria de evolução de Charles Darwin.

Estes algoritmos trabalham com uma *população* de *indivíduos* (também chamados de *cromossomos*), cada um representando uma possível solução para um dado problema. Cada indivíduo possui um valor, *aptidão*, que mede quão boa é a solução para o problema que codifica. Os indivíduos mais aptos (ou seja que possuem uma maior aptidão), têm maiores chances de se "reproduzirem" por *cruzamento*, gerando novos indivíduos, que compartilham algumas características tomadas de seus "pais".

Portanto, AGs resolvem um problema, basicamente, gerando, mudando e avaliando soluções candidatas (indivíduos) para o problema. A geração da população inicial é aleatória, as mudanças nos indivíduos são feitas pelos operadores de mutação/cruzamento, e a sua avaliação é baseada numa função que mede sua qualidade como uma solução para o problema que está sendo resolvido. Para mais informações sobre os conceitos referentes a AGs

veja [2, 6, 3, 5].

3 Modelagem

Nesta seção vamos apresentar as idéias implementadas na modelagem das fases do algoritmo genético para o problema em questão. O fluxo do AG é ilustrado na figura 1 na página seguinte.

3.1 Vizinhança entre pontos

A entrada para o problema é um mapa contendo um conjunto de pontos de demanda constituindo a área de projeto. Para obtermos informações de vizinhança entre pontos, e para facilitar as operações do AG, decidimos realizar um pré-processamento nos dados de entrada. Este pré-processamento consiste em dividir a área de projeto em uma malha de quadrados idênticos, cujo tamanho do lado deve ser especificado. Cada posição da malha, chamada *unidade*, possui um único ponto representado pelo centro de massa de todos os pontos que pertencem àquela unidade (veja figura 2 na página 5), e será a menor unidade de processamento do algoritmo. Cada unidade pode conter um ponto com demanda, ou pode ser vazia. Para maiores detalhes veja seção 3.1 de [4].

3.2 Representação

Quando usamos AGs para resolver um problema de otimização, o primeiro passo, e o mais importante, é definir a forma de representação de uma solução deste problema como um cromossomo, o qual possa ser submetido facilmente aos operadores de mutação e/ou cruzamento.

A representação escolhida é toda baseada na malha de vizinhança, conforme explicado na subseção anterior. Portanto, um cromossomo é representado por: (1) um ponteiro para a malha de vizinhança (a malha contém os pontos de demanda), (2) um vetor de seções, onde cada seção possui um conjunto de ponteiros para as unidades que estão na malha (a soma das unidades de todas as seções é igual ao número de unidades não vazias da malha de vizinhança). Com apenas estas estruturas identificamos facilmente quais são as unidades que estão numa mesma seção, bastando percorrer o vetor de seções até a posição desejada e listar as unidades pertencentes à seção. Porém, dado um ponto (ou uma unidade), em qual seção ele está? Para responder a esta pergunta facilmente, decidimos criar mais um vetor, (3), do tamanho do número de unidades não vazias da malha. Cada posição deste vetor possui dois índices: um indicando o índice da unidade (ou seja, sua posição na malha de vizinhança) e outro referente ao índice da seção a qual o ponto pertence. Um exemplo ilustrativo desta representação pode ser encontrado na figura 3 na página 6. Para maiores detalhes sobre esta representação vide documento [4].

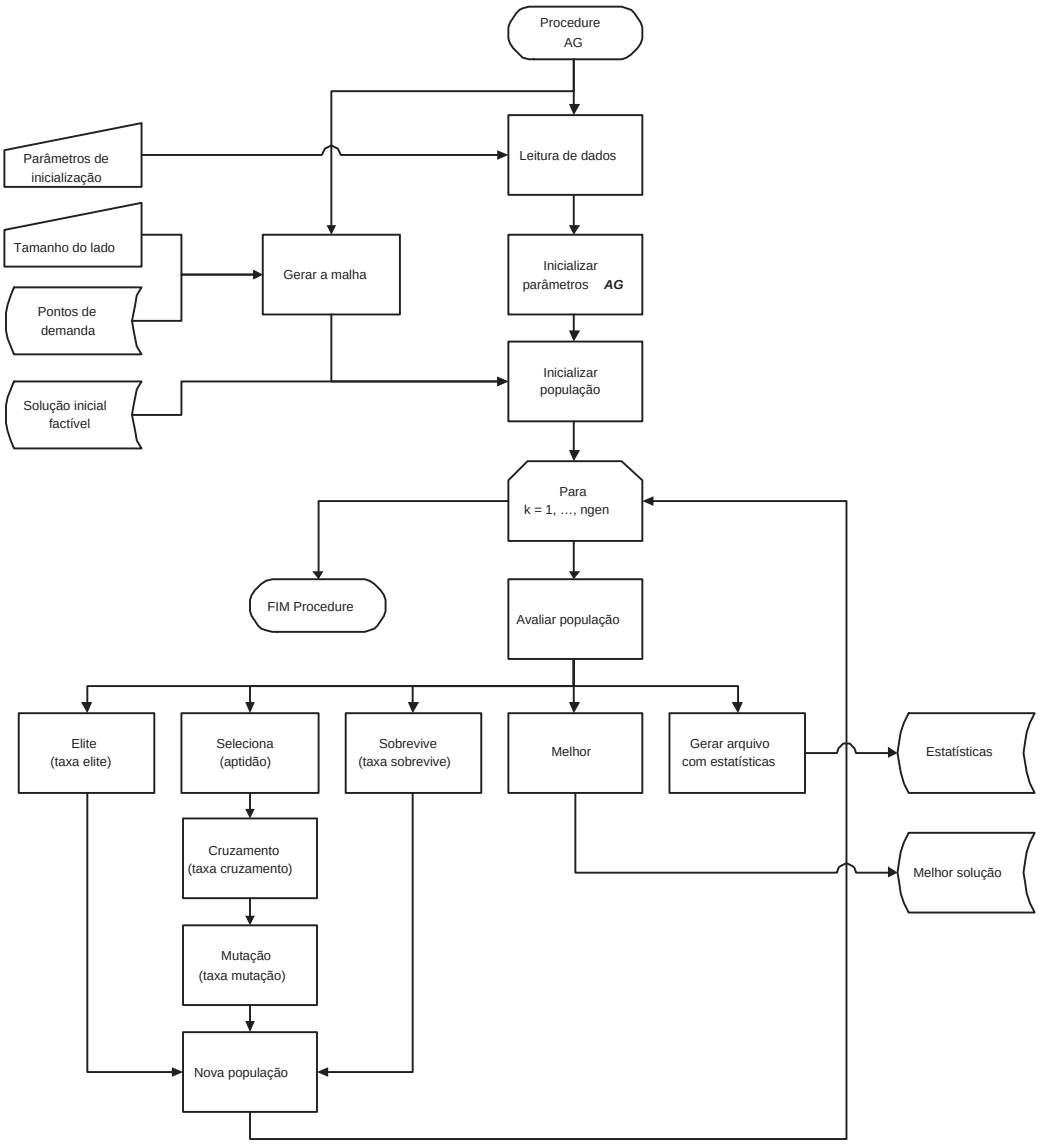
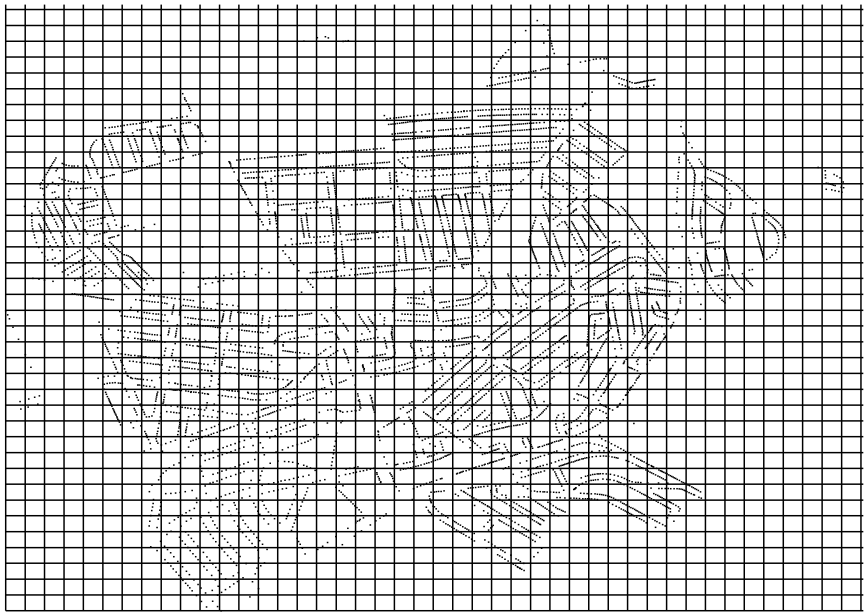
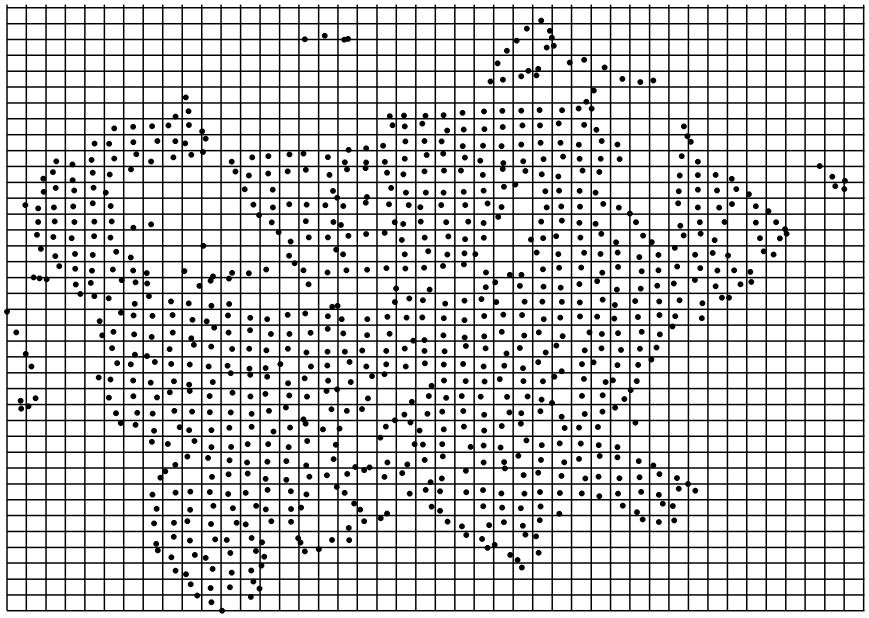


Figura 1: Fluxograma do AG.



(a) Pontos reais



(b) Pontos contraídos

Figura 2: Pré-processamento - Malha Discretizada.

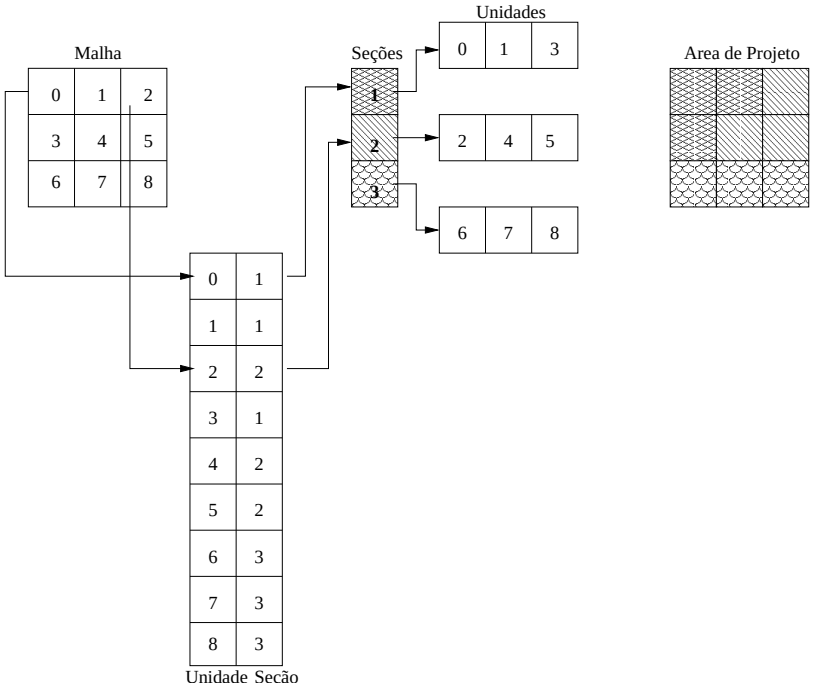


Figura 3: Exemplo de um cromossomo.

3.3 Restrições

Atualmente as restrições que estão sendo tratadas pelo algoritmo são referentes à carga (ocupação) máxima e mínima dos armários. Para o AG é difícil tratar tais restrições diretamente, de modo que não sejam aceitas soluções que as violem. Uma forma indireta de tratá-las é aplicar uma penalidade em indivíduos que estão infringindo as restrições. Como estamos interessados em minimizar o custo do projeto, podemos somar um valor $Q(p)$ ao custo dos indivíduos p. Como é muito difícil estimar este valor, realizamos testes e obtivemos o seguinte valor para $Q(p)$:

$$Q(p) = \sum_{s \in p} QS(s),$$

onde s são todas as seções da solução p e $QS(s)$ é a penalidade atribuída à seção s, definida como:

$$QS(s) = \begin{cases} c_p * (K_a \times C - D_s) & \text{se} \quad D_s < K_a \times C \\ c_p * (D_s - K_A \times C) & \text{se} \quad D_s > K_A \times C \\ 0 & \text{se} \quad K_a \times C < D_s < K_A \times C \end{cases}$$

onde $K_a \times C$ é a carga mínima do armário, $K_A \times C$ é a carga máxima do armário, D_s é a demanda total da seção s, e $c_p \in \mathbb{R}^+$ é o coeficiente da função linear da penalidade, significando que quanto maior seu valor, maior será a penalidade aplicada ao indivíduo p.

3.4 Custo da Solução

O custo da solução, $C(p)$, representada pelo cromossomo p, é calculado como:

$$C(p) = \sum_{s \in p}(c_a + f \sum_{p_i \in s}[d_i \times dist(p_a, p_i)]),$$

onde s é uma seção do cromossomo p, c_a é o custo do armário, $f = \sigma\beta$ (σ é o custo médio dos cabos por unidade de distância, β é o fator de correção da distância euclideana para a distância real), p_i é a posição do ponto i da seção s, d_i é a demanda do ponto i e p_a é a posição do armário na seção s.

3.5 Avaliação

A aptidão de um indivíduo p é calculada da seguinte forma:

$$f(p) = \frac{N - (C(p) + Q(p))}{F}$$

onde N é o menor número que torna f sempre positiva, $C(p)$ é o custo do projeto representado pela solução p, $Q(p)$ é o valor da penalidade de p e $F = \sum_{p \in P}(N - (C(p) + Q(p)))$, sendo P a população.

3.6 Seleção

Nesta fase os indivíduos mais aptos são selecionados para gerar uma nova população através do cruzamento. No método de seleção utilizado, cada indivíduo tem uma probabilidade de ser selecionado proporcional à sua aptidão. Esse método é conhecido como *Roleta (Roulette Wheel)*.

Para visualizar este método, considere uma "roleta" dividida em n regiões (tamanho da população), onde a área de cada região é proporcional à aptidão do indivíduo. Veja a figura 4 na próxima página. Após uma rodada, a posição do cursor indica o indivíduo selecionado. Este método é também denominado *amostragem universal estocástica*. Evidentemente, os indivíduos cujas regiões possuem maior área terão maior probabilidade de serem selecionados e, conseqüentemente, a seleção de indivíduos pode conter várias cópias de um mesmo indivíduo enquanto outros podem desaparecer.

3.7 Cruzamento

O cruzamento é feito da seguinte forma:

1. Selecionamos dois indivíduos para o cruzamento (*Seleção*): p_1 e p_2.

2. O ponto de cruzamento é representado por um corte na malha de vizinhança. Um corte é um conjunto de posições da malha de vizinhança m, que a divide em duas partes conexas e disjuntas.

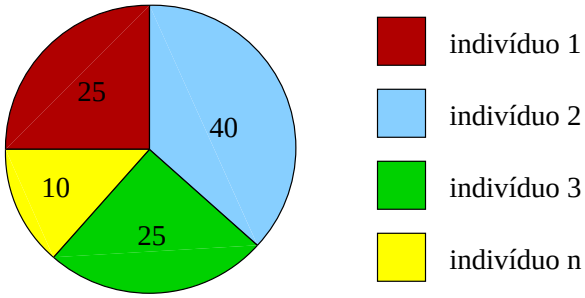


Figura 4: Seleção por roleta.

3. Realizamos vários cortes, e selecionamos aqueles dois que geram os dois melhores filhos. Um corte é considerado bom quando ao criar o filho a partir das partes de cada pai aquele tenha o menor custo. Seja o corte c_1 que divide a malha em duas partes (m_{11}, m_{12}), selecionado para criar o filho 1 e o corte $c_2 = (m_{21}, m_{22})$ selecionado para criar o filho 2.

4. Criamos o primeiro filho com as seções que resultaram de p_1 referentes aos pontos de m_{11} mais as seções resultantes de p_2 referentes aos pontos de m_{12}.

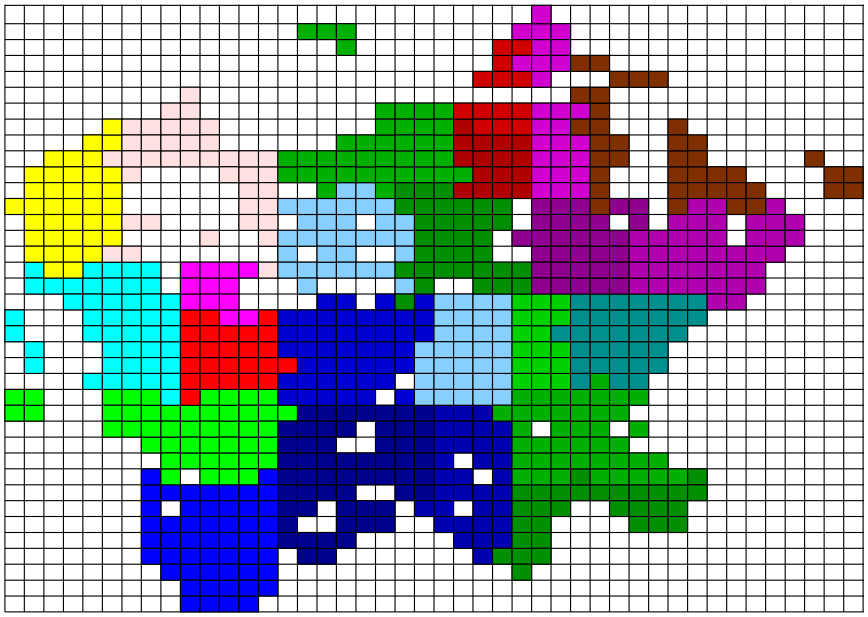
5. Criamos o segundo filho com as seções que resultaram de p_1 referentes aos pontos de m_{22} mais as seções resultantes de p_2 referentes aos pontos de m_{21}.

6. As seções resultantes do cruzamento (de ambos os filhos) que ficaram abaixo da capacidade mínima, entram num processo de diminuição de fronteira, até desaparecerem.

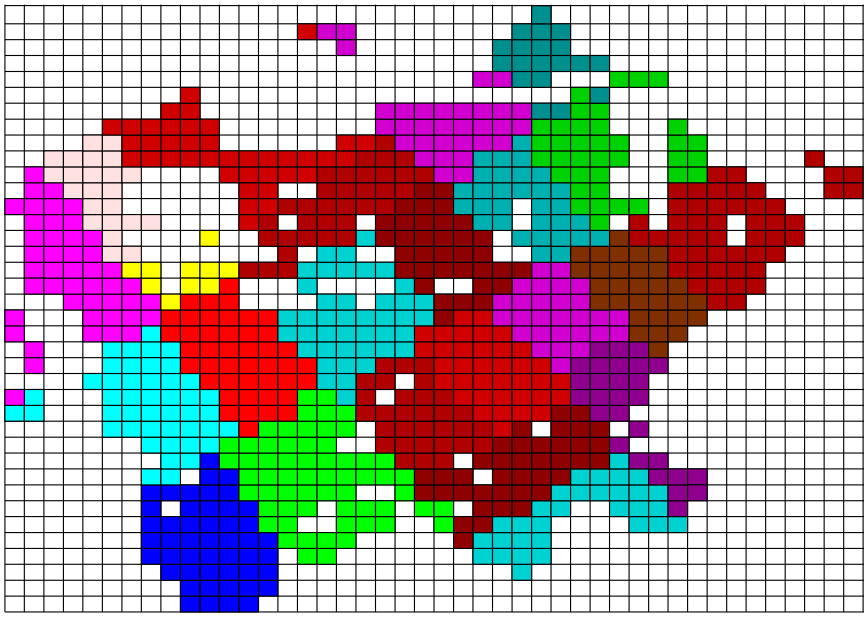
As figuras 5 e 6 ilustram um exemplo desta operação, onde as figuras 5(a) e 5(b) representam os pais selecionados para cruzamento. As posições pintadas de preto na figura 6(a) ilustram o corte feito, e as posições pintadas de amarelo ouro formam as seções que ficaram muito abaixo da demanda mínima permitida, e desaparecerão, como mostra a figura 6(b).

3.7.1 Corte

Como foi mencionado antes, um corte na malha de vizinhança é um conjunto de unidades da malha que a divide em duas partes disjuntas. Criamos quatro tipos de cortes, os quais chamamos de horizontal, vertical, sudoeste e noroeste, conforme a figura 7 na página 11. O corte é baseado na seguinte idéia: selecionamos probabilisticamente uma posição da malha, e a partir desta posição, tomamos três outras adjacentes a ela. Aplicamos uma probabilidade a cada uma destas posições, e sorteamos uma para continuar o corte. Repetimos estes passos até que uma extremidade da malha seja alcançada. Assim, de acordo com a orientação do corte que está sendo feito, dividimos a malha em duas regiões (lado 1 e lado 2). Na figura 7 na página 11 mostramos como é o comportamento do corte, quais são as posições vizinhas de acordo com sua orientação, e quais posições ficam de um mesmo lado da malha, sendo que as posições que formam o corte sempre ficam do lado 1.

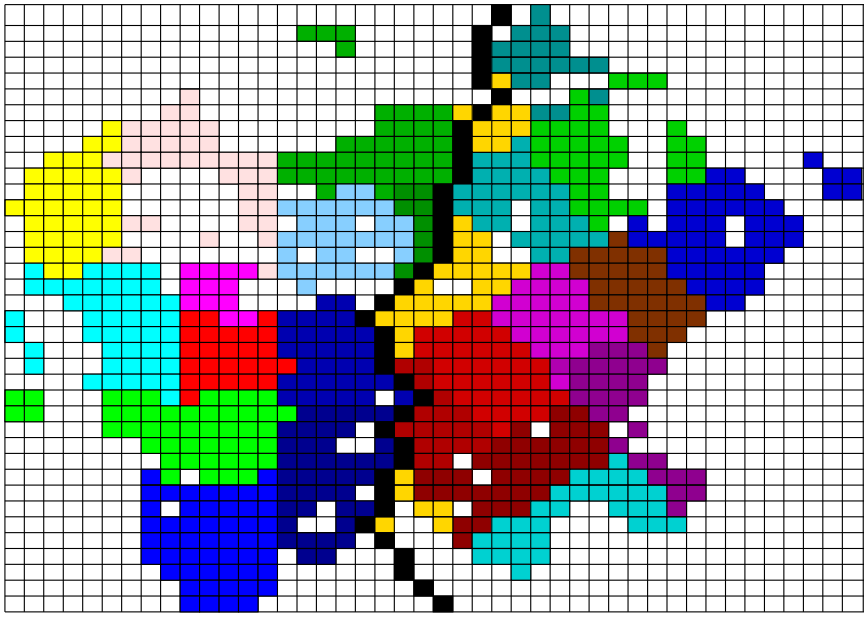


(a) Pai

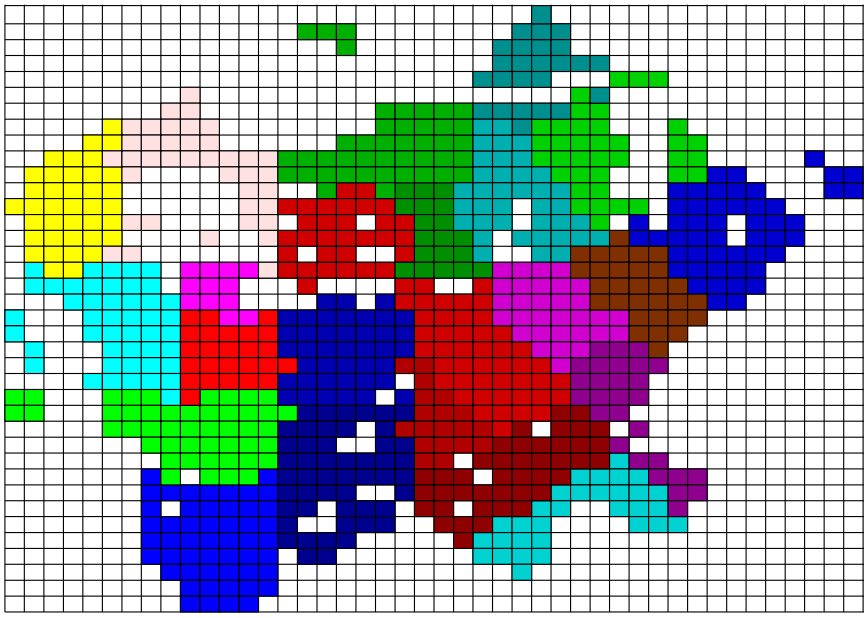


(b) Mãe

Figura 5: Pais selecionados para o cruzamento.



(a) Corte



(b) Filho

Figura 6: Corte realizado e o filho resultante do cruzamento.

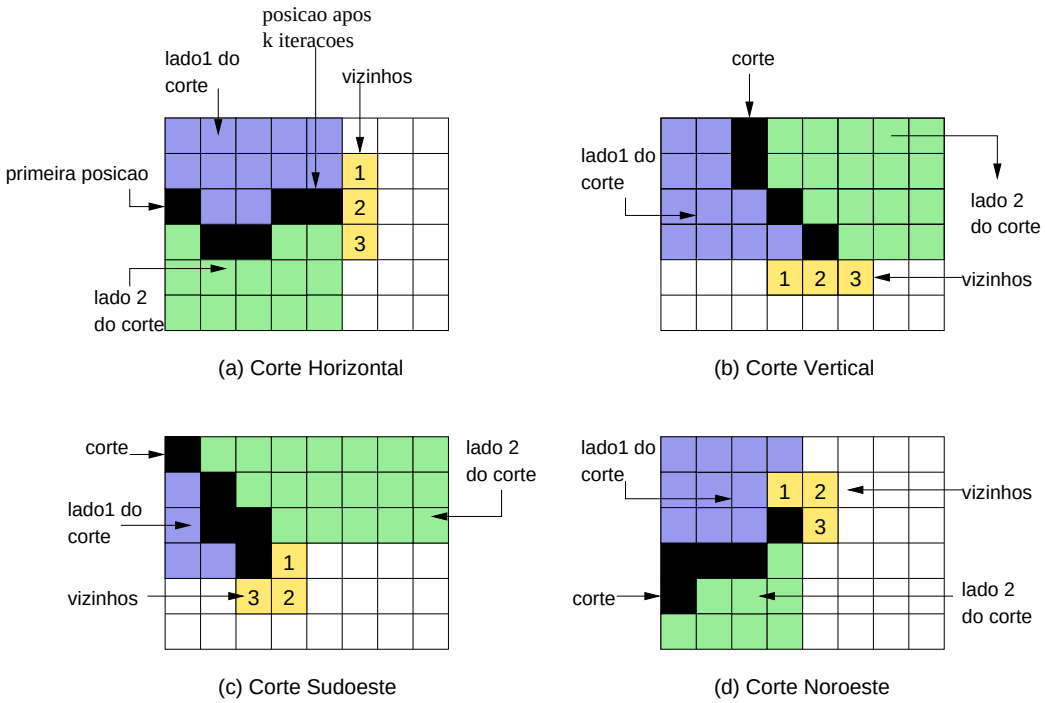


Figura 7: Os quatro tipos de corte na malha.

Para calcular a probabilidade de cada unidade ser selecionada para entrar no corte, foram utilizadas duas propriedades como segue: (1) sua posição e (2) sua seção com relação à unidade escolhida anteriormente pelo corte.

Para cada propriedade temos três casos, e a probabilidade de uma unidade é calculada pela multiplicação dos valores referentes às suas propriedades, como especificado na tabela 1. Nesta tabela, p_1, p_2 e p_3 representam as probabilidades referentes à posição da próxima unidade que pertencerá ao corte, 1, 2 ou 3, respectivamente. Os valores p_i e p_d indicam o valor da probabilidade para os casos da seção ser igual ou diferente da unidade escolhida anteriormente, e se a unidade é vazia o valor de sua probabilidade é dado por p_v.

Seção\Posição	**1**	**2**	**3**
igual	$p_1 \times p_i$	$p_2 \times p_i$	$p_3 \times p_i$
diferente	$p_1 \times p_d$	$p_2 \times p_d$	$p_3 \times p_d$
vazia	$p_1 \times p_v$	$p_2 \times p_v$	$p_3 \times p_v$

Tabela 1: Probabilidades para seleção da unidade que entra no corte.

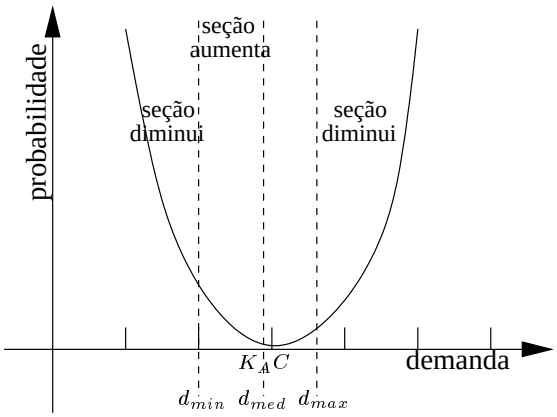


Figura 8: Função de probabilidade para mutação.

3.8 Mutação

Nesta versão do algoritmo genético testamos dois operadores de mutação. O primeiro deles é baseado na demanda das seções do indivíduo p selecionado para sofrer mutação. Escolhemos probabilisticamente uma seção s da solução p para ser alterada. Se esta demanda está acima de um valor d_{max} ou abaixo de um valor d_{min}, esta seção sofrerá uma diminuição de fronteira, e se sua demanda estiver entre d_{min} e d_{med} ela sofrerá um aumento de fronteira. Ou seja, a seção s doará ou tomará pontos das seções vizinhas conforme sua demanda. A função de probabilidade para escolher a seção que sofrerá mutação é uma parábola com uma única raiz dada pela carga máxima do armário ($K_A \times C$), e está ilustrada na figura 8. Os valores d_{min}, d_{med} e d_{max} são parâmetros do algoritmo.

Como tanto o operador de cruzamento como o de mutação, descritos anteriormente levam em consideração apenas a demanda das seções, foi criado um novo operador de mutação, o qual é aplicado nos indivíduos que já estão no limite mínimo do número de seções (calculado pela demanda total do projeto divido pela carga máxima do armário), com o intuito de minimizar o custo de cabeamento. Este operador funciona da seguinte forma:

- Seleciona uma seção s, do cromossomo. A seleção é feita pela roleta (4.8) onde a probabilidade de cada seção ser selecionada é proporcional ao seu custo de cabeamento.
- Troca as unidades da fronteira para as seções vizinhas. Esta troca é feita se a unidade está mais próxima do armário da seção vizinha do que do armário da seção s, e se o armário da seção vizinha ainda tem espaço para atender a esta unidade.

4 Implementação

Esta seção descreve as principais classes implementadas para o algoritmo genético. Cada classe é apresentada em uma subseção que contém uma breve descrição de seu propósito,

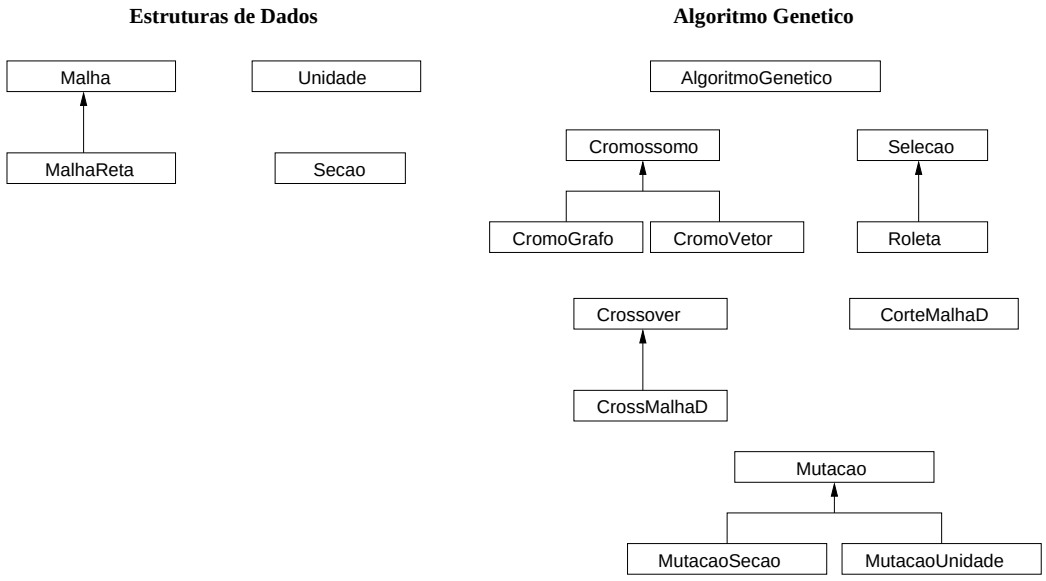


Figura 9: Hierarquia de classes.

bem como detalhes dos seus principais atributos e métodos.

A figura 9 ilustra as classes descritas nesta seção e suas hierarquias.

4.1 Classe Unidade

Esta classe implementa uma posição da malha de vizinhança.

4.1.1 Principais Atributos

- **m_ponto** $\rightarrow$ este atributo guarda as coordenadas (x, y), do centro de massa de todos os pontos que ela representa, como também a sua *demanda* total.
- **m_n** $\rightarrow$ número de pontos reais que estão na unidade.
- **m_id** $\rightarrow$ identificador da unidade, que também é utilizado para saber a sua posição na malha de vizinhança.

4.1.2 Principais Métodos

- **bool ehVazia()** $\rightarrow$ retorna *true* se a unidade não possui pontos, ou seja, se é vazia, e *false* caso contrário.

4.2 Classe Malha

Esta é uma classe abstrata que representa uma malha de vizinhança colocada sobre os pontos de demanda.

4.2.1 Principais Atributos

- **m_ptos** $\rightarrow$ pontos de demanda, sobre os quais a malha irá ser colocada.
- **m_locais** $\rightarrow$ pontos com demanda superior a um valor (K_U), que são considerados seções de serviço locais. Estes pontos são retirados do cálculo de seções, pois eles por si só formam uma ou mais seções.

4.2.2 Principais Métodos

- **void vizinhos(Unidade*** u**, VetorDeUnidades&** $unds$**)** $\rightarrow$ é um método virtual, que deve ser implementado nas classes filhas. Este método, aplica a propriedade da malha para encontrar as unidades vizinhas de u.

4.3 Classe MalhaReta

É derivada da classe *Malha* (seção 4.2 na página anterior). Implementa a malha de vizinhança descrita na seção 3.1 na página 3. Ela é formada por uma matriz de linhas e colunas, onde cada posição é um objeto da classe *Unidade* (seção 4.1 na página anterior).

4.3.1 Principais Atributos

- **m_pontos** $\rightarrow$ conjunto de pontos de demanda reais.
- **m_lado** $\rightarrow$ tamanho do lado da malha, é a partir deste valor que as dimensões da malha são calculadas.
- **m_linhas** $\rightarrow$ número de linhas da malha.
- **m_colunas** $\rightarrow$ número de colunas da malha.

4.3.2 Principais Métodos

- **void vizinhos(Unidade*** u**, VetorDeUnidades&** $unds$**)** $\rightarrow$ retorna as unidades vizinhas da unidade u.

4.4 Classe Secao

Esta classe implementa uma seção de serviço, ou seja um conjunto de pontos de demanda, com uma posição para o armário de distribuição.

4.4.1 Principais Atributos

- **m_id** $\rightarrow$ identificador da seção.
- **m_armario** $\rightarrow$ ponto que representa a posição do armário.

- **m_demanda** → demanda total que a seção atende.
- **m_unidades** → vetor de *Unidade* (seção 4.1 na página 13), indicando os pontos que pertencem a esta seção.

4.4.2 Principais Métodos

- **calculaPosicaoArmario()** → calcula a posição do armário, que é dada pelo centro de massa dos pontos pertencentes a seção.
- **calculaPenalidade()** → calcula a penalidade da seção, se esta fere as restrições de carga máxima ou mínima.

4.5 Classe Cromossomo

O cromossomo é uma classe base virtual e não pode ser instanciada. Ela define um número de constantes e protótipos de funções específicas para o cromossomo e suas classes derivadas. O tipo é usado para especificar qual o tipo de cromossomo (qual representação está sendo utilizada). Nesta classe está armazenada o ponteiro para a malha de vizinhança, e também o vetor que identifica qual unidade da malha está em uma seção e vice-versa (vide figura 3 na página 6).

4.5.1 Principais Atributos

- **m_malha** → ponteiro para a malha de vizinhança.
- **m_pontosSecoes** → vetor de dois inteiros contendo o índice da unidade na malha e o índice da seção que ele pertence.
- **m_custo** → custo do cromossomo.
- **m_penalidade** → penalidade do cromossomo, calculada como a soma das penalidades de suas seções.

4.5.2 Principais Métodos

- **void secoesVizinhas(Unidade*** u**, VetorDeSecoes &** $secs$**)** → este é um método virtual puro que tem como objetivo retornar as seções vizinhas da unidade u.
- **double calculaCusto()** → calcula o custo total do projeto representado pelo cromossomo. Realiza a soma dos custos de cada seção pertencente a ele.
- **double calculaPenalidade()** → calcula a penalidade da solução representada pelo cromossomo. Realiza a soma das penalidades de cada seção pertencente a ele.

4.6 Classe CromoVetor

Esta classe é derivada da classe *Cromossomo* (seção 4.5 na página precedente) e implementa um cromossomo como um vetor de seções.

Outra idéia seria representar o cromossomo como um grafo de vizinhança entre seções, assim o método que retorna as seções vizinhas seria mais eficiente e transparente.

4.6.1 Principais Atributos

- **m_secoes** $\rightarrow$ vetor de seções (seção 4.4 na página 14), que formam a solução representada pelo cromossomo.

4.6.2 Principais Métodos

- **void secoesVizinhas(Unidade*** u**, VetorDeSecoes &** $secs$**)** $\rightarrow$ Retorna as seções vizinhas da unidade u. Este método é útil durante a operação de mutação. Para saber quais são as seções vizinhas de uma unidade, é necessário analisar as seções das unidades vizinhas de u (através do método *vizinhos()* da malha de vizinhança.

4.7 Classe Selecao

Esquemas de seleção são usados para selecionar cromossomos de um população para casamento. A classe *Selecao* define o comportamento básico de um seletor. É uma classe abstrata e não pode ser instanciada. O método *seleciona* recebe um vetor de pesos (ou aptidão de cada cromossomo) e retorna o índice do peso selecionado. Temos portanto que relacionar o índice do peso com o índice do cromossomo selecionado na população.

4.7.1 Principais Métodos

- **int seleciona(const VetorDePesos&** $pesos$**)** $\rightarrow$ retorna um índice do vetor $pesos$, correspondente à posição selecionada, é um método virtual puro e deve ser implementado pelas classes derivadas.

4.8 Classe Roleta

Esta classe é derivada da classe *Selecao* e implementa o método de seleção descrito na seção 3.6 na página 7.

4.8.1 Principais Métodos

- **int seleciona(const VetorDePesos&** $pesos$**)** $\rightarrow$ retorna um índice do vetor $pesos$, correspondente à posição selecionada.

4.9 Classe Crossover

Esta é uma classe abstrata, não pode ser instanciada. Ela realiza a operação de cruzamento. Para cada tipo de representação deve ser implementada uma classe que saiba cruzar os cromossomos específicos.

4.9.1 Principais Métodos

- **void cruza(int** k**, Cromossomo*** p_1**, Cromossomo*** p_2**, Cromossomo*&** f_1**, Cromossomo*&** f_2**)** $\rightarrow$ é um método virtual que deve ser sobrecarregado pelas classes filhas. Este método recebe os pais do cruzamento (p_1 e p_2), e retorna os filhos resultantes (f_1 e f_2).

4.10 Classe CrossMalhaD

Esta classe implementa a operação de cruzamento sob a MalhaReta (seção 4.2 na página 13), e é derivada da classe *Crossover*.

4.10.1 Principais Métodos

- **void cruza(int** k**, Cromossomo*** p_1**, Cromossomo*** p_2**, Cromossomo*&** f_1**, Cromossomo*&** f_2**)** $\rightarrow$ seleciona o melhor ponto de cruzamento (corte 4.11), dentre os k realizados, sendo que o melhor corte é aquele que minimiza o custo das seções resultantes nos filhos. Este método invoca o método *criaCorte* da classe *Corte*, seleciona o melhor e chama o método *criaFilho*, explicado a seguir.

- **void criaFilho(const VetorDeUnidades&** $corte1$**, const VetorDeUnidades&** $corte2$**, Cromossomo*** p_1**, Cromossomo*** p_2**, Cromossomo*&** f**)** $\rightarrow$ cria o cromossomo filho f, a partir das seções resultantes do p_1 com os pontos do $corte1$, e das seções do p_2 com os pontos do $corte2$.

4.11 Classe CorteMalhaD

Esta classe calcula um ponto de cruzamento para ser utilizado na operação de cruzamento sob a *MalhaReta* (seção 4.2 na página 13). Os pontos de cruzamento implementados nesta classe são os ilustrados na figura 7 na página 11.

4.11.1 Principais Métodos

- **void criaCorte(double** p_1**, double** p_2**, double** p_3**, double** p_i**, double** p_v**, double** p_d**, Cromossomo*** c**, VetorDeUnidades &** $corte1$**, VetorDeUnidades &** $corte2$**)** $\rightarrow$ seleciona aleatoriamente um dentre os quatro tipos de corte (figura 7 na página 11) para ser feito na malha do cromossomo c. Recebe como parâmetro as probabilidades que guiam a direção do corte (p_1, p_2, p_3, p_i, p_v, p_d), e retorna em $corte1$ as unidades que ficaram de um lado do corte e em $corte2$ as que ficaram do outro lado.

4.12 Classe Mutacao

Esta é uma classe base virtual que não deve ser instanciada. Ela representa a operação de mutação. O método desta classe que realiza mutação é uma função que recebe um ponteiro para um cromossomo genérico, e o número de mutações que serão realizadas nele.

4.12.1 Principais Métodos

- **void muda(Cromossomo *& c, int $numMutacoes$)** $\rightarrow$ este é um método virtual puro que é invocado pelo algoritmo genético. Ele recebe um cromossomo c para ser mutado e o número de mutações a serem feitas, $numMutacoes$. Deve ser implementado nas classes derivadas.

4.13 Classe MutacaoSecao

Esta classe implementa a mutação macro, ou seja, que altera as seções, aumentando ou diminuindo suas fronteiras. Conforme explicado na seção 3.8 na página 12.

4.13.1 Principais Métodos

- **void muda(Cromossomo *& c, int $numMutacoes$)** $\rightarrow$ este método é invocado pelo algoritmo genético. Ele recebe um cromossomo c para ser mutado e irá realizar $numMutacoes$ mudanças nele, ou seja, selecionará $numMutacoes$ seções para aumentar ou diminuir a fronteira.
- **void diminuiFronteira(Cromossomo*& c, Secao*& s, double $quantidade$)** $\rightarrow$ diminui a fronteira da seção s até que $s.demanda() \leq s.demanda() - quantidade$. Esta diminuição de fronteira é feita doando pontos de fronteira para as seções vizinhas de s.
- **void aumentaFronteira(Cromossomo*& c, Secao*& s, double $quantidade$)** $\rightarrow$ aumenta a fronteira da seção s até que $s.demanda() \geq s.demanda() + quantidade$. Este aumento é feito agrupando pontos de fronteira das seções vizinhas.

4.14 Classe MutacaoUnidade

Esta classe implementa a micro mutação, ou seja, altera as unidades de fronteira das seções de serviço. Esta mutação consiste em percorrer a fronteira de uma seção e caso haja uma seção vizinha tal que a distância do armário desta à unidade de fronteira é menor do que a distância do armário de sua seção, esta unidade migra para a seção vizinha.

4.14.1 Principais Métodos

- **void muda(Cromossomo *& c, int $numMutacoes$)** $\rightarrow$ este método é invocado pelo algoritmo genético. Ele recebe um cromossomo c para ser mutado e irá realizar $numMutacoes$ mudanças nele, ou seja, selecionará proporcionalmente ao custo $numMutacoes$

seções para trocar os pontos de fronteira. Isto é, dada uma seção s, percorremos cada unidade u da sua fronteira, e caso haja uma seção vizinha s_v a sua fronteira tal que $s_v.armario().dist(u) < s.armario().dist(u)$, trocamos a unidade u de seção e ela passa a pertencer a seção s_v.

4.15 Classe AlgoritmoGenetico

Esta classe implementa um algoritmo genético simples. Quando criamos um algoritmo genético, devemos especificar o tamanho da população, a quantidade de indivíduos que se reproduzirão e sofrerão mutação. Também podemos especificar uma taxa de elitismo, ou seja, uma quantidade dos melhores indivíduos que sempre sobreviverão nas gerações futuras. Os operadores genéticos utilizados também devem ser indicados, na criação do algoritmo genético.

O critério de parada para este algoritmo é o número de gerações, e deve ser indicado no método que realiza a evolução. As estatísticas, como o melhor indivíduo por geração, a média dos custos por geração, demanda mínima, máxima e média do melhor cromossomo e da média da população são escritos em arquivos com nomes pré-definidos, num diretório indicado pelo usuário.

O método de evolução do algoritmo genético primeiro chama a função de iniciação, e inicia todos os cromossomos da população. Então avalia a população, realiza os operadores de seleção e cruzamento, depois o de mutação, seleciona a elite, e então cria a nova população. O algoritmo realiza este ciclo até que o número de gerações seja atingido.

4.15.1 Principais Atributos

- **m_nPop** → número de indivíduos da população.
- **m_populacao** → vetor de ponteiros para a classe Cromossomo (seção 4.5 na página 15), representa a população que sofrerá a evolução.
- **m_tipo** → tipo do cromossomo, ou seja qual classe ele pertence, para instanciar a população.
- **m_melhor** → guarda a melhor solução encontrada.
- **m_selecao** → ponteiro para a classe Selecao (seção 4.7 na página 16), que implementa o método de seleção que será aplicado.
- **m_cruzamento** → ponteiro para a classe Crossover (seção 4.9 na página 17), que implementa o método de cruzamento que será aplicado.
- **m_mutacao** → ponteiro para a classe Mutacao (seção 4.12 na página anterior), que implementa o método de mutacao que será aplicado.
- **m_numCruzamento** → número de indivíduos que sofrerão cruzamento, este número é calculado por $m\_nPop * t_c$, onde t_c é a taxa de cruzamento passada como parâmetro no construtor do AG.

- **m_numElite** $\rightarrow$ número de indivíduos da elite que sobreviverão durante as gerações, este número é calculado por $m\_nPop * t_e$, onde t_e é a taxa de elite passada como parâmetro no construtor do AG.

- **m_numSobrevive** $\rightarrow$ número de indivíduos da população corrente que sobreviverão sem sofrer alguma operação, este número é calculado por $m\_nPop - (m\_numCruzamento + m\_numElite)$.

- **m_numMutacao** $\rightarrow$ número de indivíduos que sofreram cruzamento e que irão ser mudados, este número é calculado por $m\_numCruzamento * t_m$, onde t_m é a taxa de mutação passada como parâmetro no construtor do AG.

4.15.2 Principais Métodos

- **bool iniciaPopulacao(Malha*** m**)** $\rightarrow$ inicia todos os cromossomos da população e atualiza o ponteiro da malha de cada um com o parâmetro m. O algoritmo da solução inicial utilizado é o descrito no documento [1].

- **void atualizaPopulacao()** $\rightarrow$ este método calcula os custos dos indivíduos da população que tiveram sua solução alterada.

- **void geraPesosPopulacao(VetorDePesos&** $pesos$**)** $\rightarrow$ calcula a aptidão de cada indivíduo e coloca seu valor no vetor $pesos$. Este vetor será enviado para o método $seleciona$ da classe Selecao (seção 4.7 na página 16).

- **bool run(Malha*** m**, int** $ngen$**)** $\rightarrow$ este é o método que realiza a evolução do algoritmo genético. Ele chama os métodos *inicializaPopulacao, atualizaPopulacao, m_selecao* $\rightarrow$*seleciona, m_cruzamento*$\rightarrow$*cruza, m_mutacao*$\rightarrow$*muda* $ngen$ vezes.

5 Resultados

5.1 Ambiente Computacional

Para os testes realizados nesta seção foi utilizado um processador Intel Pentium 4 2 GHz com 1024MB de memória RAM.

5.2 Parâmetros

Realizamos uma variedade de testes para ajustar os parâmetros recebidos pelo algoritmo genético e pelas classes listadas na seção 4 na página 12. Os testes apresentados nesta seção foram realizados para a região Sul da Vila Mariana. Existem dois conjuntos de parâmetros, os referentes ao projeto de seção de serviço e os referentes ao algoritmo genético, listados na tabela 2 na página oposta e na tabela 3 na página 22, respectivamente.

Para medir a qualidade de uma solução estamos levando em consideração o custo total do projeto e a distribuição de demanda entre as seções.

Dados do Projeto		
Símbolo	**Descrição**	**Valor**
σ	custo médio dos cabos por unidade de distância	R$ 0,0323
β	fator de correção da distância euclideana para a real	1,9
c_a	custo do armário	R$ 230.000,00
C	capacidade do armário	600
K_A	fator de carga máxima no armário	80%
K_a	fator de carga mínima no armário	30%
K_U	demanda localizada mínima para caracterizar seção de serviço local	300

Tabela 2: Dados do projeto.

O único parâmetro da tabela 2 que foi alterado durante os testes, foi o K_U (seção de serviço local). Este valor influencia fortemente no custo do projeto visto que para cada ponto com demanda maior ou igual a K_U um ou mais armários de distribuição deverão ser instalados para atendê-lo, e o custo do armário é muito elevado. Além disso, também influencia na distribuição de demanda entre as seções, pois removendo estes pontos temos menos demanda para alocar em armários, podendo resultar numa demanda média baixa em relação à carga máxima do armário.

5.2.1 Plano de Testes

Nesta seção apresentamos os testes realizados e seus resultados. Como o algoritmo genético é uma heurística com escolhas probabilísticas, cada teste foi realizado cinco vezes, e o resultado apresentado é a média das cinco evoluções.

Demanda Localizada Mínima de 240

A tabela 4 na próxima página apresenta os parâmetros dos testes realizados levando em consideração os dados da tabela 2 com $K_U = 240$. Para estes testes temos três pontos que se tornaram seção de serviço local, com demandas iguais a 1.200, $303{,}6$ e 264. Estes pontos não serão considerados durante a evolução do AG. A demanda total do projeto que o AG deve tratar é $9.848{,}79$.

Neste conjunto de testes, variamos as taxas de cruzamento e mutação (a variação da taxa de cruzamento influencia nas taxas de elite e sobrevivência), para sabermos quão sensíveis são estes parâmetros para o algoritmo (testes: 1, 2, 3 e 4). Uma outra abordagem nos testes foi variar as probabilidades utilizadas durante a operação de corte, juntamente com uma diminuição do coeficiente utilizado na função de penalidade (teste 5).

Na tabela 5 na página 24 apresentamos os resultados para os testes da tabela 4 na página seguinte. As características que ressaltamos das soluções são o número total de seções do projeto, o custo total, o custo aproximado de cabeamento, a porcentagem do custo de cabo em relação ao custo total, a demanda máxima, mínima e média das seções do projeto e o desvio padrão desta distribuição. O custo total do projeto apresentado na tabela 5 não

Dados do AG		
Símbolo	**Descrição**	**Referência**
$lado$	tamanho do lado da malha de vizinhança	seção 4.3 na página 14
$ngen$	números de gerações do AG	seção 4.15 na página 19
$npop$	tamanho da população	seção 4.15 na página 19
t_c	taxa de cruzamento	seção 4.15 na página 19
t_e	taxa de elite	seção 4.15 na página 19
t_s	taxa de sobreviventes	seção 4.15 na página 19
t_m	taxa de mutação	seção 4.15 na página 19
c_p	coeficiente da função de penalidade	seção 3.3 na página 6
p_1	probabilidade para o vizinho 1 durante o corte	tabela 1 na página 11
p_2	probabilidade para o vizinho 2 durante o corte	tabela 1 na página 11
p_3	probabilidade para o vizinho 3 durante o corte	tabela 1 na página 11
p_i	probabilidade se a unidade vizinha for da mesma seção	tabela 1 na página 11
p_d	probabilidade se a unidade vizinha for de seção diferente	tabela 1 na página 11
p_v	probabilidade se a unidade vizinha for vazia	tabela 1 na página 11

Tabela 3: Dados do algoritmo genético.

Teste	$lado$	$ngen$	$npop$	t_e %	t_c %	t_s %	t_m %	p_1 %	p_2 %	p_3 %	p_i %	p_d %	p_v %	c_p
1	20	500	50	10	80	10	20	25	25	50	40	10	50	800
2	20	1000	50	40	50	10	20	25	25	50	40	10	50	800
3	20	500	50	10	80	10	1	25	25	50	40	10	50	800
4	20	1000	50	10	80	10	10	25	25	50	40	10	50	800
5	20	1000	50	10	80	10	40	35	35	30	20	10	70	650

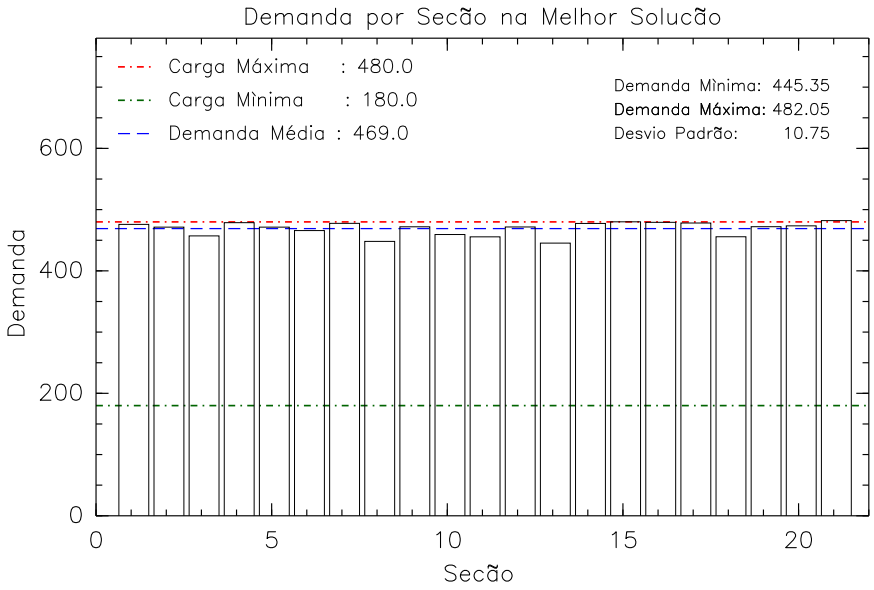
Tabela 4: Plano de teste para $K_U = 240$.

contém o custo relativo às seções de serviço locais listadas acima, que resultam em um custo equivalente a cinco armários de distribuição.

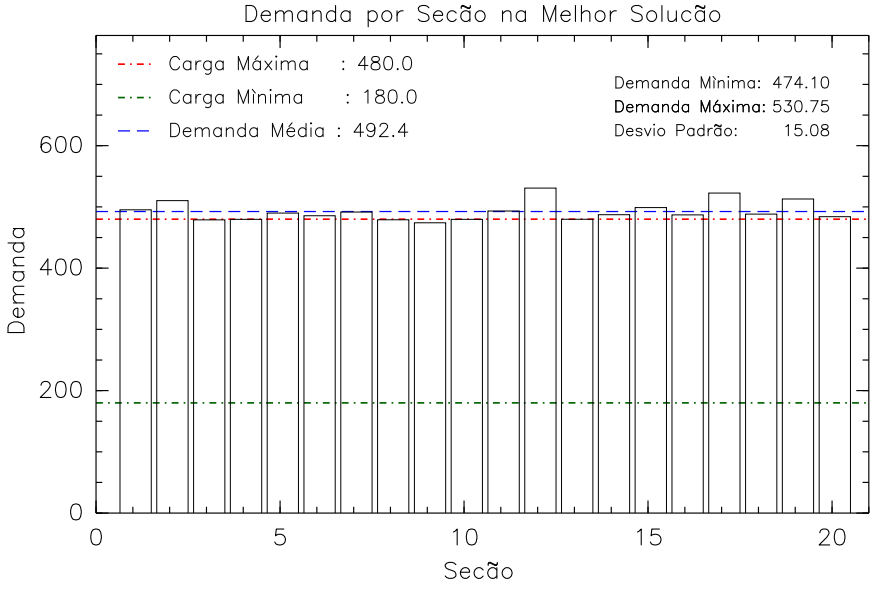
Das tabelas 4 e 5 na página 24, concluímos que a diferença de custo entre os testes de 1 a 4 é muito pequena, variando no máximo de $R\$3.000,00$, relativo a variação no custo de cabo. Apenas quando alteramos o coeficiente da função de penalidade (teste 5), damos uma maior liberdade para aumentar a carga de um armário, assim o AG conseguiu diminuir uma seção, porém a demanda máxima quase chegou ao limite físico do armário $(537,95)$, e a demanda média ultrapassou a carga máxima do mesmo $(492,4)$. As figuras 10(a) e 10(b) ilustram a distribuição de demanda dos testes 2 e 5 respectivamente.

Demanda Localizada Mínima de 300

Realizamos alguns testes com demanda localizada mínima igual a 300, concluímos que esta é a melhor configuração para a região testada, visto que a demanda média das seções $(481{,}56)$ ficou quase igual a carga máxima desejada, o desvio padrão da distribuição foi



(a) Teste 2



(b) Teste 5

Figura 10: Distribuição de demanda.

Teste	tempo (min)	número de seções	custo total	custo cabos	% cabo	Demanda			
						desvio	mínima	máxima	média
1	19	21	4.914.211,23	84.211,23	1,71	8,87	444,00	483,30	468,99
2	24	21	4.911.437,67	81.437,61	1,66	11,60	439,00	484,75	468,99
3	19	21	4.913.105,19	83.105,19	1,69	12,80	441,00	489,55	468,99
4	35	21	4.911.251,64	81.251,64	1,65	11,80	436,00	484,68	468,99
5	39	20	4.684.910,53	84.910,53	1,81	17,10	470,00	537,95	492,40

Tabela 5: Resultado dos testes para $K_U = 240$.

pequeno e possui uma seção local a menos, resultando num custo total somado com os custos das seções locais menor do que o custo com $K_U = 240$.

A tabela 6 apresenta os testes realizados levando em consideração os dados da tabela 2 na página 21 com $K_U = 300$. Para estes casos de teste existem dois pontos considerados seções de serviço local, com demandas 1.200 e $303,6$. A demanda total agora é $10.112,8$.

Teste	$lado$	$ngen$	$npop$	t_e %	t_c %	t_s %	t_m %	p_1 %	p_2 %	p_3 %	p_i %	p_d %	p_v %	c_p
1	50	10000	50	10	80	10	20	25	25	50	40	10	50	800
2	20	1000	50	10	80	10	0	25	25	50	40	10	50	800
3	20	2000	50	10	80	10	10	25	25	50	40	10	50	800
4	20	1000	50	10	80	10	10	35	35	30	20	10	70	800
5	20	500	50	10	80	10	40	25	25	50	40	10	50	650
6	20	500	50	20	80	0	20	25	25	50	40	10	50	500
7	20	1000	50	40	50	10	20	25	25	50	40	10	50	800

Tabela 6: Plano de teste para $K_U = 300$.

Teste	tempo (min)	número de seções	custo total	custo cabos	% cabo	Demanda			
						desvio	mínima	máxima	média
1	90	21	4.911.302,51	81.302,51	1,66	4,88	471,80	492,00	481,56
2	34	22	5.141.387,80	81.393,59	1,58	27,26	336,95	476,27	459,68
3	60	21	4.912.139,14	82.139,14	1,67	3,17	473,70	490,15	481,56
4	41	21	4.914.177,60	84.287,73	1,72	3,91	472,43	490,90	481,56
5	37	21	4.912.520,68	82.520,68	1,68	7,45	468,00	499,95	481,56
6	37	20	4.682.336,94	92.336,94	1,99	27,90	472,00	578,40	505,60
7	29	21	4.912.028,30	82.028,30	1,67	4,99	471,20	499,80	481.56

Tabela 7: Resultado dos testes para $K_U = 300$.

A tabela 7 apresenta os resultados para os testes da tabela 6. Podemos concluir que:

- Quanto maior o número de gerações mais chances o algoritmo tem para procurar soluções melhores, porém o tempo de processamento aumenta, chegando a um ponto

onde o algoritmo não evolui mais, como podemos ver no gráfico da figura 11 (teste 1).

- O tamanho do lado da malha influencia no tempo de processamento. Quanto menor ele for, mais tempo o algoritmo gasta. Concluímos isto analisando o teste 1 com lado de malha igual a 50, cujo tempo foi de 90 minutos para 10.000 iterações, enquanto o teste 3 com lado igual a 20 demorou 60 minutos para 2.000 iterações.

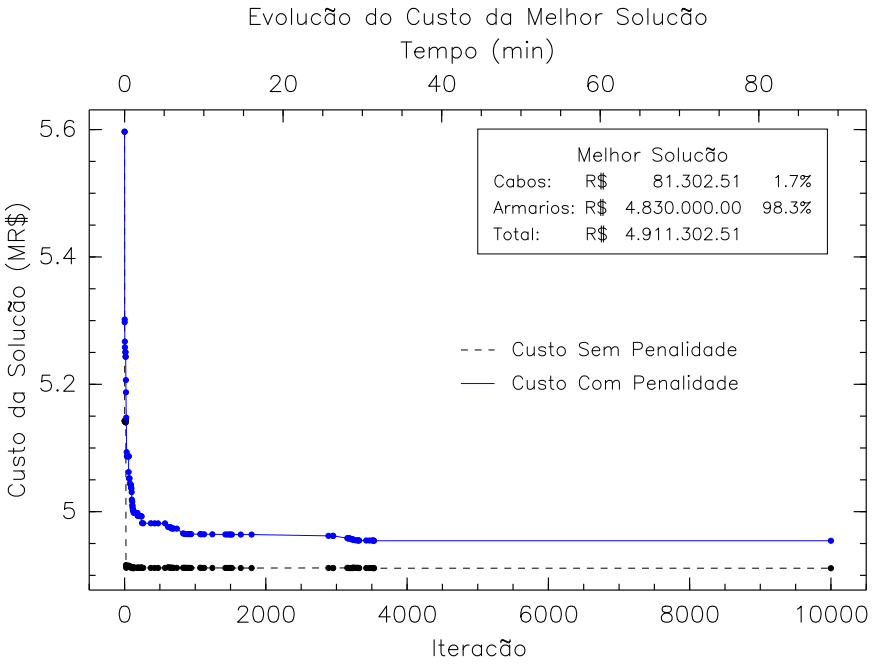


Figura 11: Evolução do genético no teste 1.

- Sem taxa de mutação o algoritmo não conseguiu diminuir uma seção, resultando numa média de demanda por seção baixa e um custo mais elevado (teste 2).
- Variando a taxa de mutação entre 10% e 40% o custo permanece quase constante (testes 3 e 5).
- Variando a taxa de elite entre 10% e 40% o custo permanece quase constante (testes 3 e 7).
- A variação das probabilidades de corte influencia no custo, e resultou numa diferença de aproximadamente R$ 2.000,00 a mais do que a melhor. Cabe salientar que esta diferença é do custo do cabeamento que está sendo aproximado, pois ainda não utilizamos o arruamento (teste 4).
- Diminuindo o coeficiente da penalidade, o algoritmo tem maior liberdade para sobrecarregar um armário, e neste caso conseguiu diminuir uma seção, aumentando a

demanda média por seção (teste 6). Nas figuras 12(a) na página oposta e 12(b) na próxima página ilustramos a distribuição de demanda para o teste 3 (com distribuição mais uniforme e perto de $K_A \times C = 480$) e o teste 6 (com demanda média muito alta, mas com uma seção a menos), respectivamente. Portanto este parâmetro parece ser bastante útil. Quando utilizado em regiões onde já se sabe que a estimativa de crescimento é baixa, pode-se colocar um valor baixo, caso contrário o parâmetro deve ter um valor mais alto.

5.3 Solução

Nesta seção apresentamos os gráficos das soluções usando os parâmetros do teste 7 da tabela 6 na página 24, aplicados às regiões Norte, Leste e Sul da Vila Mariana e ao Bairro da Liberdade.

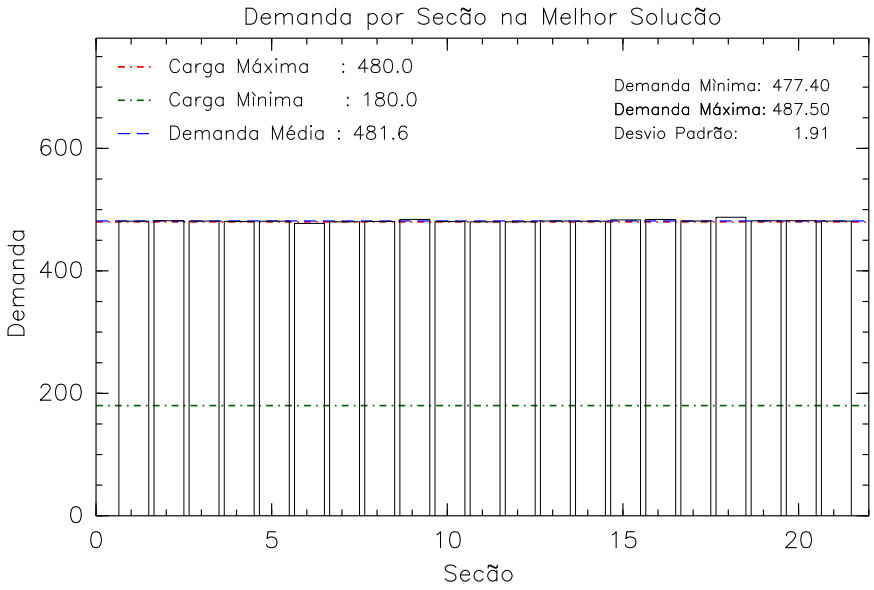
5.3.1 Região Sul da Vila Mariana

As figuras desta seção retratam os gráficos da solução resultante da aplicação do algoritmo genético para a região Sul da Vila Mariana.

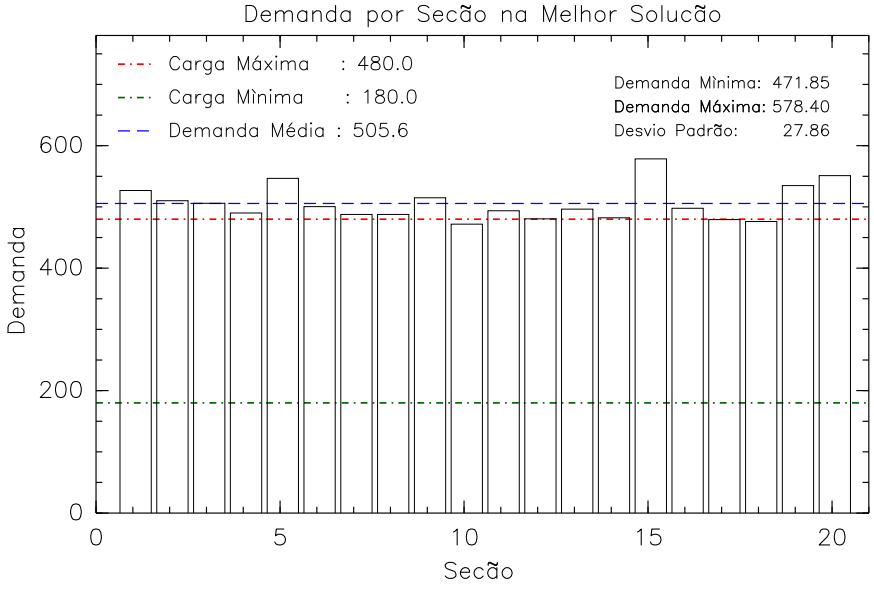
A figura 13 na página 28 apresenta a distribuição de demanda entre as seções da solução. A partir desta figura observamos que a solução possui o número mínimo de armários, que é dado pela demanda total dividida pela carga máxima do armário ($\frac{10.112{,}8}{480} = 21{,}07$), e estes armários estão todos equilibrados, bem perto da carga máxima permitida. A média da distribuição de demanda entre as seções é igual a $481{,}6$, e o desvio padrão desta distribuição é igual a $4{,}15$, ou seja, todas as seções possuem exatamente a demanda desejada que foi passada como parâmetro para o algoritmo ($C \times K_A = 480$). Não temos nenhuma seção abaixo da carga mínima (180), e nenhuma muito perto da capacidade do armário (600), sendo que a seção que possui a menor demanda atende $472,95$ pontos (muito próximo da carga máxima do armário). Resultando em seções uniformes que atendem a demanda projetada, e possuem espaço físico para possíveis alterações e aumento da rede.

A figura 14 na página 29 apresenta o custo de cabeamento em cada seção. Observamos que algumas seções possuem um custo muito maior do que outras, devido à extensão da seção. Quanto maior for sua área de atendimento, mais alto será o custo de atender os pontos que a ela pertencem. O número de seções é inversamente proporcional ao custo de cabo, ou seja, quanto maior o número de seções menor será o custo de cabeamento, podendo até reduzir o custo de cabeamento a zero, se o número de seções for igual ao número de pontos de demanda.

Podemos observar, também, que o custo de cabo é muito inferior ao custo dos armários, portanto o algoritmo tenta diminuir ao máximo o número de seções, para então tentar distribuir os pontos de demanda entre as seções de forma que o custo do cabo diminua. Mais uma vez salientamos que os custos ilustrados na figura 14, são custos estimados, calculados pela distância do ponto ao armário que o atende, ponderado pela sua demanda. Ao aplicar esta solução na realidade, estes custos irão certamente modificar, visto que a posição do armário nem sempre poderá ser àquela proposta pelo algoritmo.



(a) Teste 3



(b) Teste 6

Figura 12: Distribuição de demanda com $K_U = 300$.

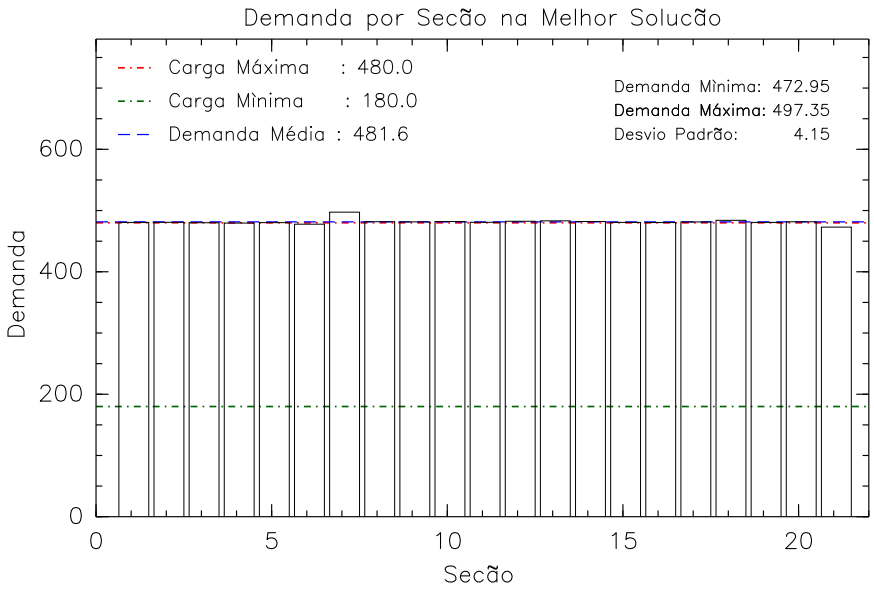


Figura 13: Solução para o Sul da Vila Mariana: demanda por seção.

A figura 15 na página 30 ilustra a evolução do algoritmo genético, ou seja, indica o custo total da melhor solução por iteração. A linha em preto representa o custo da melhor solução e a linha em azul ilustra o custo mais a penalidade. Os pontos mostram quando o algoritmo conseguiu uma solução melhor. A queda brusca do custo que observamos nas primeiras gerações é devido à diminuição de um armário (cujo custo é muito alto). Observamos também, que após a diminuição do armário, o algoritmo não possui uma melhora significativa, porém, é nesta fase que os pontos de demanda são distribuídos entre as seções, para que o custo do cabo diminua. Nesta figura, apresentamos também o custo total (4.910.908,46), o custo de armários (4.830.000,00) e o custo de cabeamento (80.908,46) da melhor solução encontrada.

Na figura 16 na página 30 a linha em preto ilustra custo do cabo por geração e a linha em azul ilustra a quantidade de seções (armários) por geração. Esta figura representa o que acontece com o algoritmo nas iterações seguintes à diminuição de um armário, o que não é possível de observar pela figura 15. Observamos que o ao diminuir um armário o custo de cabo aumenta consideravelmente, isto porque algumas seções tiveram que atender pontos mais distantes de seus armários, e durante a evolução do AG, o custo de cabo conseguiu reduzir sendo menor do que o custo de cabo quando a solução possuía uma seção a mais (primeiras iterações).

A figura 17 na página 31 mostra o particionamento dos pontos de demanda em seções de serviço.

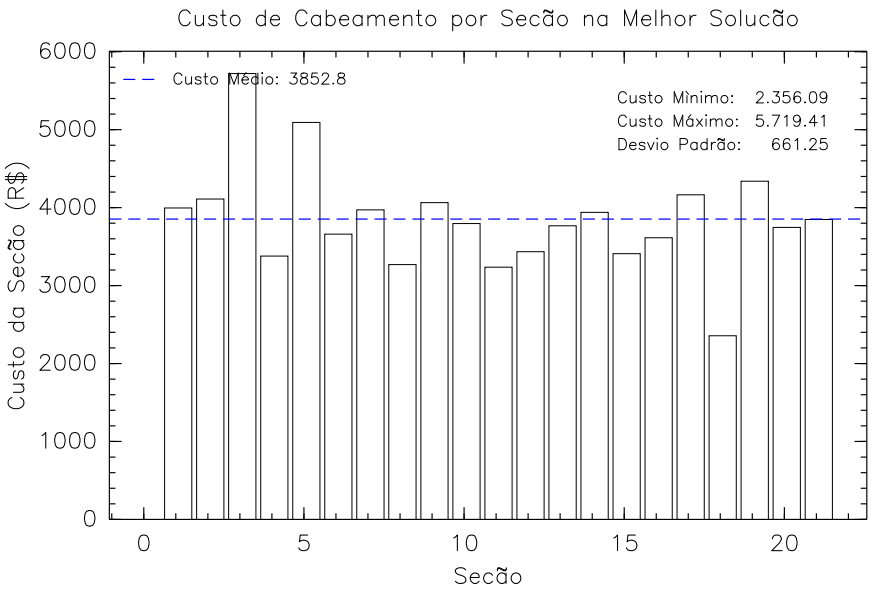


Figura 14: Solução para o Sul da Vila Mariana: custo por seção.

5.3.2 Região Leste da Vila Mariana

As figuras desta seção ilustram os gráficos resultantes da aplicação do algoritmo genético para a região Leste da Vila Mariana.

A figura 18 apresenta a distribuição de demanda entre as seções resultantes da solução. A demanda total desta região é $11.349{,}6$, portanto o número mínimo de armários é igual a 24. É o que obtemos nesta solução. A média da distribuição de demanda entre as seções é igual a $472{,}9$, e o desvio padrão desta distribuição é igual a $4{,}15$, ou seja, as seções estão muito próximas da carga máxima desejada com uma variação muito pequena entre as demandas de cada uma. Como consideremos a distribuição de demanda entre as seções como medida de qualidade de uma solução, podemos concluir que a solução apresentada possui uma alta qualidade.

A figura 19 apresenta o custo de cabeamento em cada seção. Observamos que algumas seções possuem um custo muito maior do que outras, devido à extensão da seção. Quanto maior for sua área de atendimento, mais alto será o custo de atender os pontos que a ela pertencem.

A figura 20 na página 34 ilustra a evolução do algoritmo genético durante as gerações, ou seja, indica o custo total da melhor solução por iteração. A linha em preto representa o custo da melhor solução e a linha em azul ilustra este custo mais a penalidade. Observamos que a penalidade aplicada às melhores soluções é pequena, isto indica que existem poucas seções que estão acima da carga máxima permitida, e a quantidade ultrapassada é pequena. A queda brusca no custo que observamos nas primeiras gerações é devido à diminuição de um armário. Nesta figura, apresentamos também o custo total ($5.607.074{,}85$), o custo de armários ($5.520.000{,}00$) e o custo de cabeamento ($87.074{,}85$) da melhor solução encontrada.

Na figura 21 na página 34 a linha em preto ilustra custo do cabo por geração e a linha em

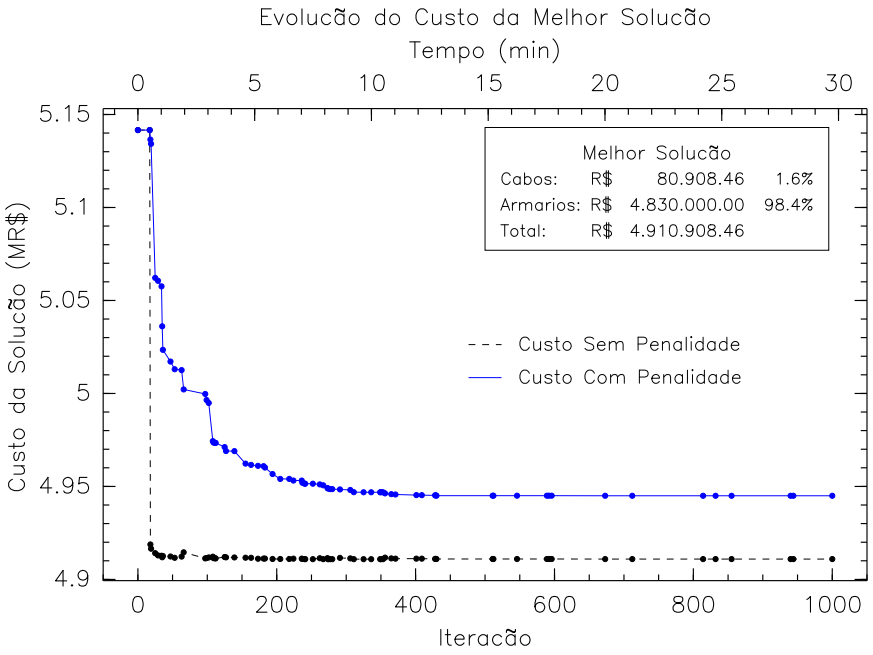


Figura 15: Solução para o Sul da Vila Mariana: custo total por iteração.

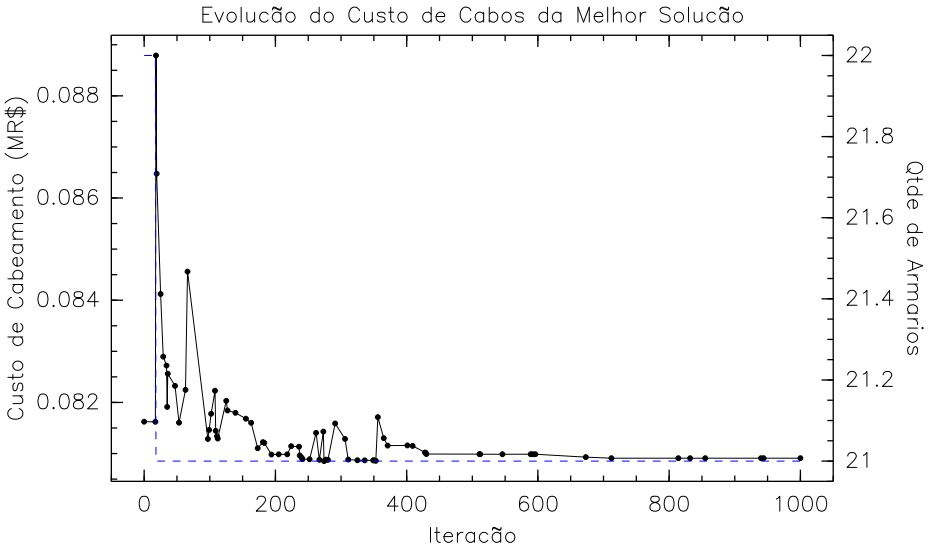


Figura 16: Solução para o Sul da Vila Mariana: custo de cabeamento por iteração.

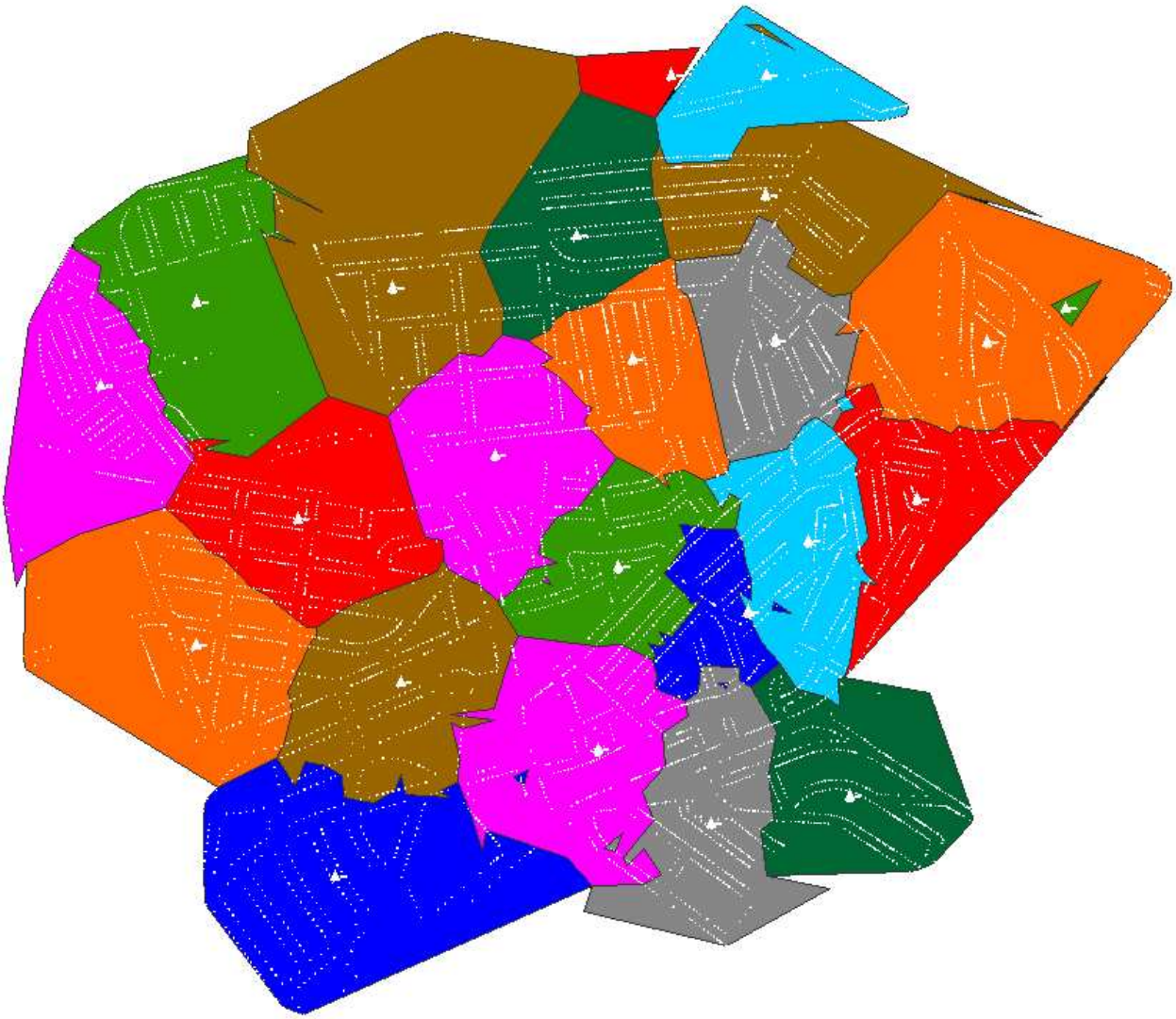


Figura 17: Solução para o Sul da Vila Mariana.

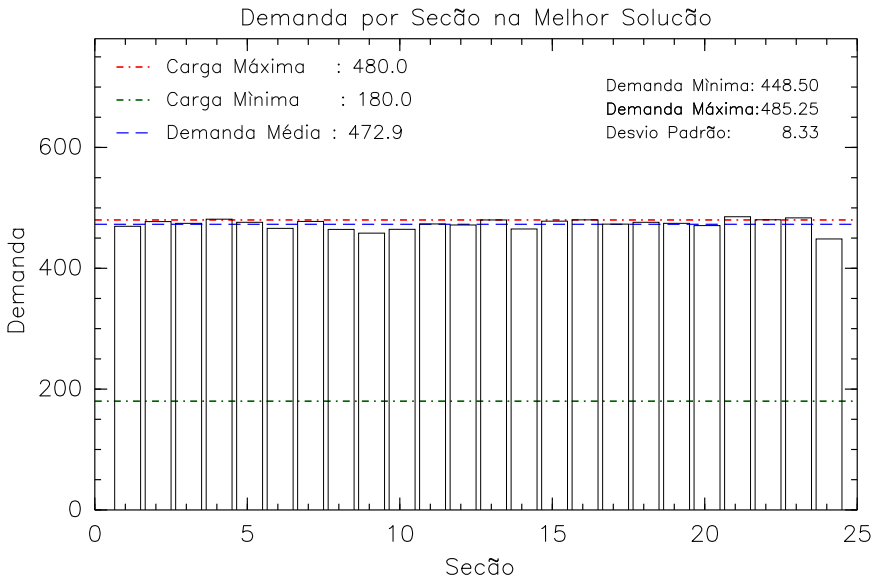


Figura 18: Solução para o Leste da Vila Mariana: demanda por seção.

azul ilustra a quantidade de seções por geração. A primeira iteração representa a solução gerada pelo algoritmo de solução inicial, este resultou numa solução com 25 armários e um custo de cabeamento elevado. No final da execução do algoritmo conseguimos uma solução com 24 armários e um custo de cabeamento inferior ao inicial.

A figura 22 na página 35 mostra o particionamento dos pontos de demanda em seções de serviço.

5.3.3 Região Norte da Vila Mariana

A figura 23 na página 36 ilustra a distribuição de demanda nas seções da melhor solução encontrada pelo AG, para os parâmetros do caso 7 da tabela 6 na página 24. Nesta figura observamos que a demanda média das seções é igual a $476{,}1$ e possui 72 seções. A distribuição de demanda está bastante uniforme com um desvio padrão de $9{,}03$.

As figuras 24 e 25 na página 37 ilustram a evolução do algoritmo genético quanto ao custo total e custo de cabeamento da melhor solução da população, respectivamente.

A figura 26 na página 38 mostra o particionamento dos pontos de demanda em seções de serviço.

Aumentando a taxa de mutação para 80%, o algoritmo diminuiu o número de seções para 71 e a média de demanda por seção subiu para $482{,}8$, resultando numa solução com um custo bem inferior ao custo ilustrado anteriormente. A distribuição de demanda para este caso pode ser observada na figura 27 na página 39, onde constatamos que a maior demanda de uma seção é $529{,}30$ e a menor é $463{,}50$ e o desvio padrão da distribuição de demanda entre as seções é de $10{,}46$.

A figura 28 na página 39 ilustra o custo de cabeamento de cada seção da solução. A figura 29 ilustra a evolução do algoritmo genético quanto ao custo total da melhor solução

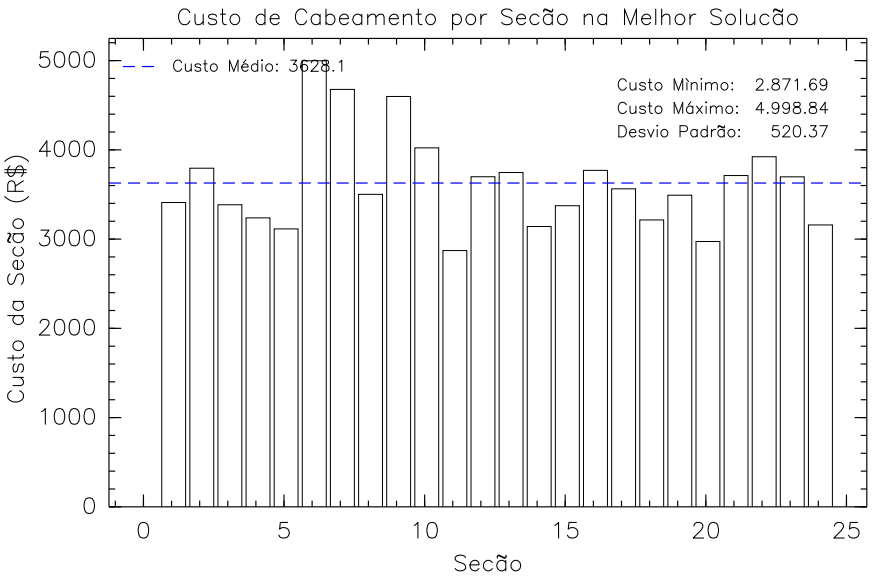


Figura 19: Solução para o Leste da Vila Mariana: custo por seção.

por iteração. Nas primeiras iterações o número de seções da melhor solução oscila entre 71 e 72, devido à penalidade aplicada às seções que estão acima da carga máxima.

A figura 30 apresenta a evolução do custo de cabo da melhor solução por iteração, nesta figura observamos que ao tentar distribuir os pontos de demanda nas seções, o custo de cabo aumenta e diminui, conforme o número de armários oscila, até que o algoritmo consegue uma configuração dos pontos tal que o custo do cabo diminua e o número de armários fique mínimo.

A figura 31 na página 41 apresenta a divisão da área de projeto em seções de serviço para o teste com taxa de mutação igual a 80%.

5.3.4 Bairro da Liberdade

Assim como para a Vila Mariana, vários testes foram realizados para o Bairro da Liberdade. Esta instância é muito maior que a anterior, visto que possui muitos pontos e com demandas altas bem concentradas.

A tabela 8 na página 36 indica os parâmetros de dois testes realizados para o Bairro da Liberdade que iremos detalhar nesta seção. O teste 1 resultou numa solução melhor geometricamente, porém o teste 2 resultou numa solução com três armários a menos e portanto num custo bem inferior, como podemos comparar pela tabela 9 na página 36.

As figuras listadas a seguir apresentam a solução do algoritmo para o conjunto de parâmetros do teste 2 da tabela 8.

A figura 32 na página 42 ilustra a distribuição de demanda entre as seções da solução, nela observamos que a demanda média é de $481{,}4$ muito próxima da carga máxima desejada. A seção com demanda máxima atende $560{,}05$ pontos de demanda e a com demanda mínima atende $411{,}10$ pontos. O desvio padrão desta distribuição é $21{,}46$. Esta instância é

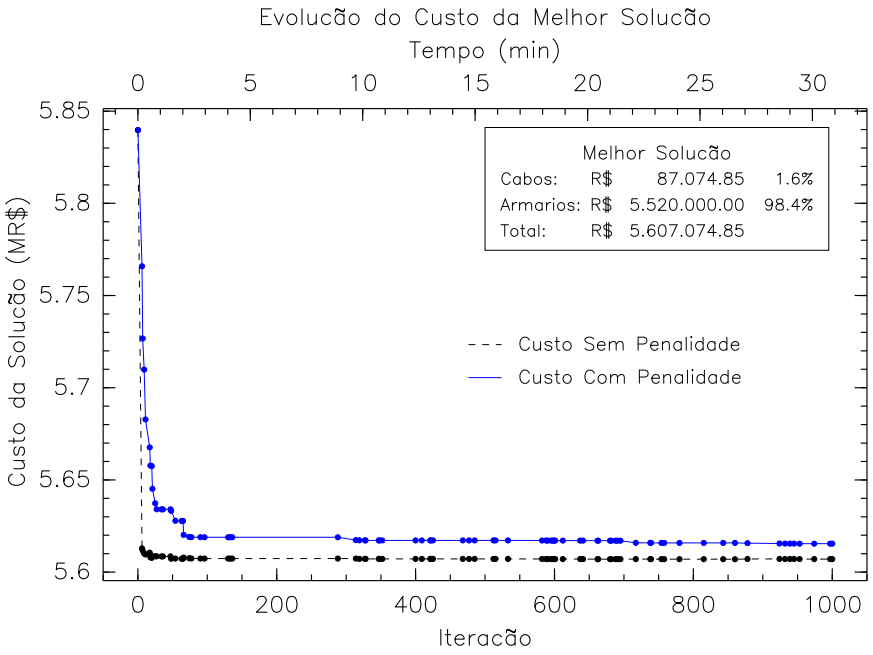


Figura 20: Solução para o Leste da Vila Mariana: custo total por iteração.

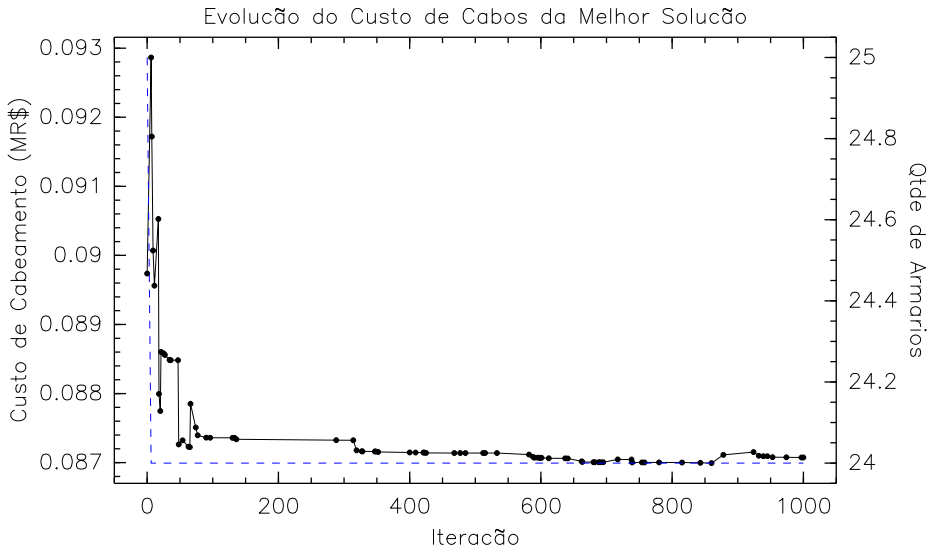


Figura 21: Solução para o Leste da Vila Mariana: custo de cabeamento por iteração.

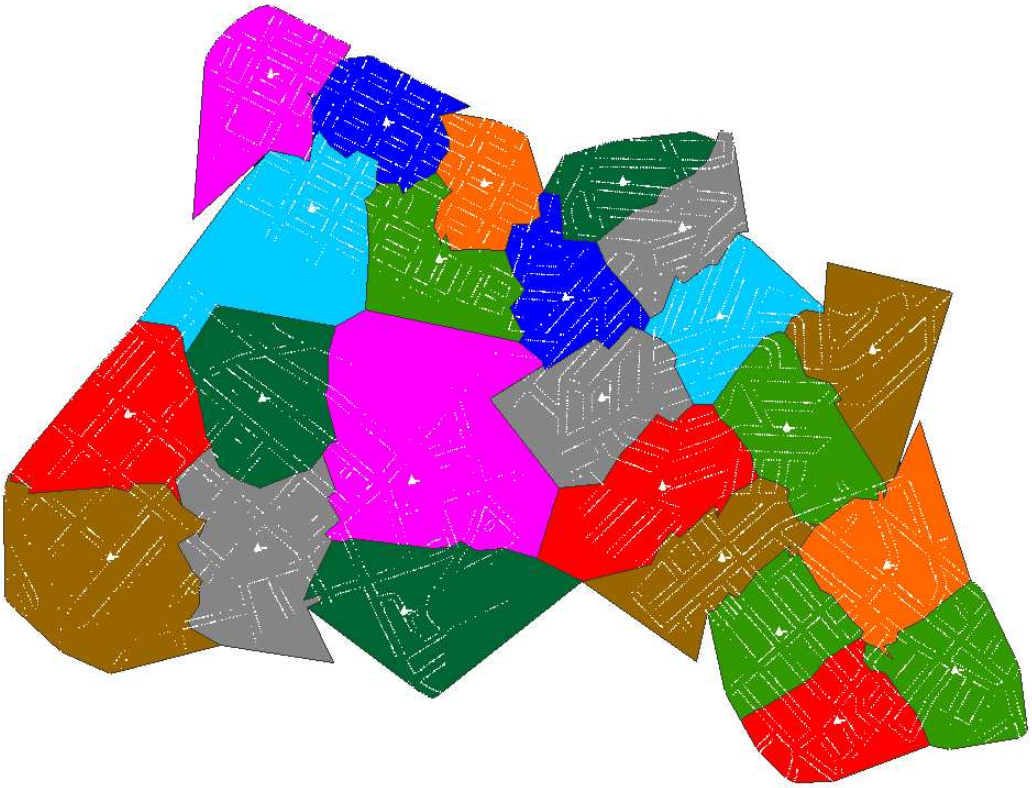


Figura 22: Solução para o Leste da Vila Mariana.

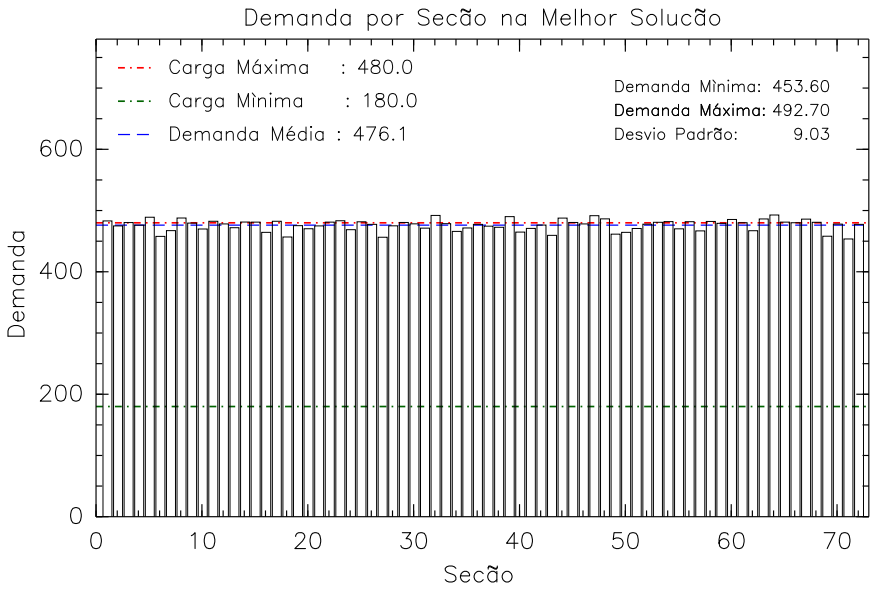


Figura 23: Solução para o Norte da Vila Mariana: demanda por seção

Teste	$lado$	$ngen$	$npop$	t_e %	t_c %	t_s %	t_m %	p_1 %	p_2 %	p_3 %	p_i %	p_d %	p_v %	c_p
1	20	1000	50	10	80	10	20	25	25	50	40	10	50	1000
2	20	1000	50	10	80	10	40	25	25	50	40	10	50	650

Tabela 8: Testes para o Bairro da Liberdade.

muito maior, densa e complexa do que as anteriores, pois possui muito mais seções e pontos de demanda concentrados. Dado estes fatores, concluímos que o resultado encontrado pelo algoritmo foi adequado.

A figura 33 na página 42 mostra o custo de cabeamento das seções, onde o menor custo é igual a $R\$150,00$ o que indica que existe uma seção com poucos pontos de demanda alta. Por outro lado, existem seções com muitos pontos dispersos, o que resulta num custo elevado de cabeamento.

Pela figura 34 na página 44 acompanhamos a evolução do custo da melhor solução durante as gerações do algoritmo. Percebemos várias quedas no gráfico referentes às quedas nos custos de cabo e armários, principalmente. O algoritmo proporcionou uma melhora de

Teste	**tempo (min)**	**número de seções**	**custo total**	**custo cabos**	**% cabo**	**Demanda**			
						desvio	**mínima**	**máxima**	**média**
1	550	190	44.363.631,95	663.631,95	1,5	19,34	423,00	525,30	473,80
2	600	187	43.661.223,22	651.223,28	1,5	21,46	411,10	560,05	481,40

Tabela 9: Resultado dos testes para o Bairro da Liberdade.

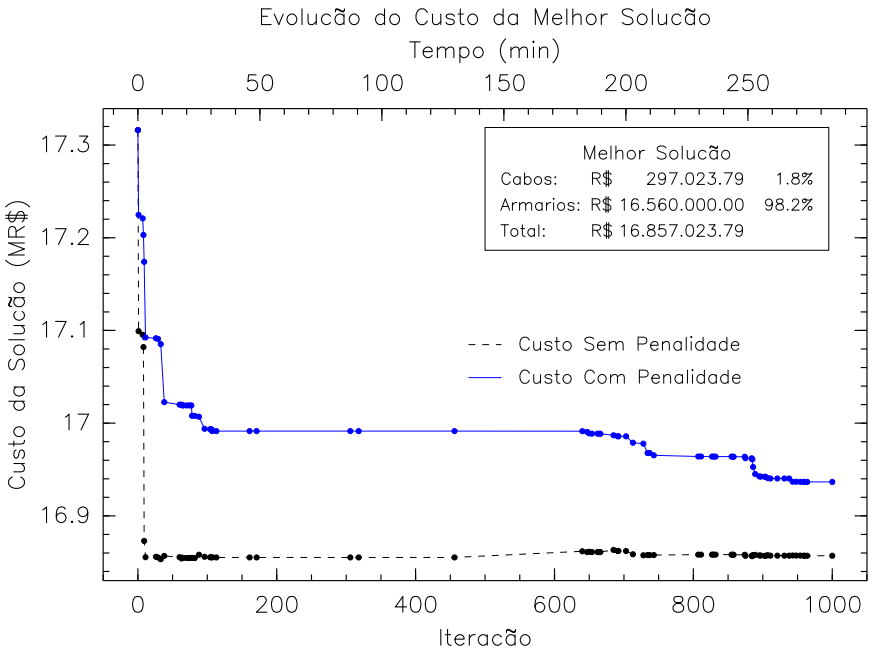


Figura 24: Solução para o Norte da Vila Mariana: custo total por iteração

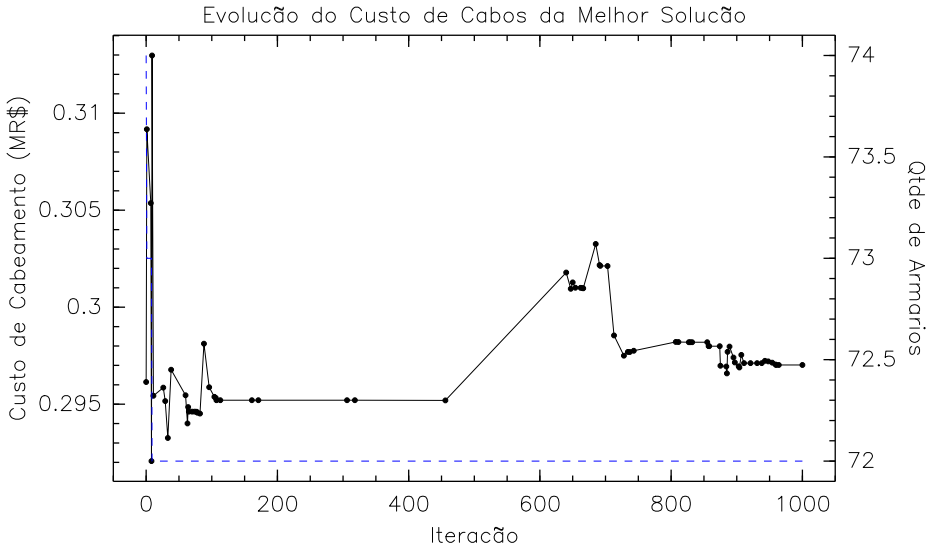


Figura 25: Solução para o Norte da Vila Mariana: custo de cabeamento por iteração

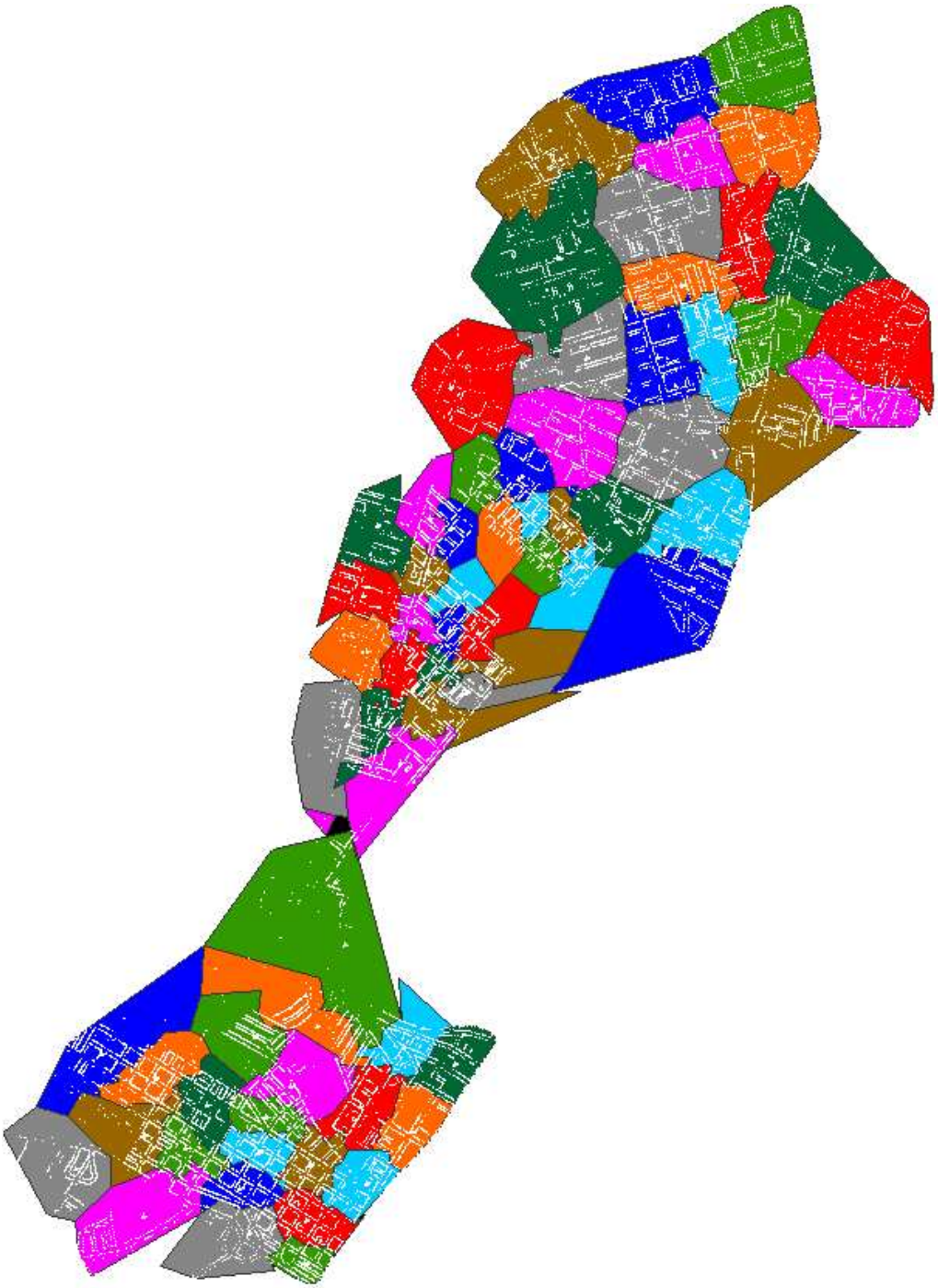


Figura 26: Solução para o Norte da Vila Mariana.

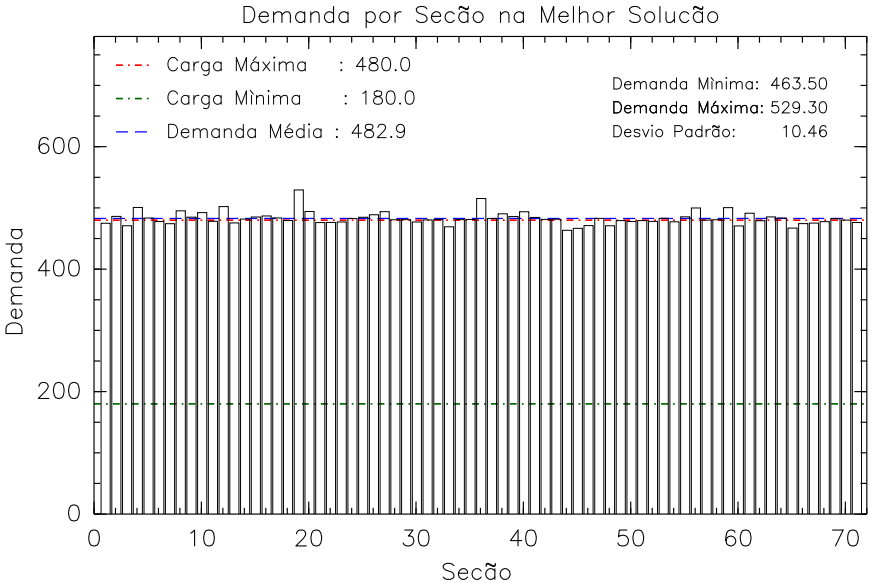


Figura 27: Solução para o Norte da Vila Mariana: demanda por seção

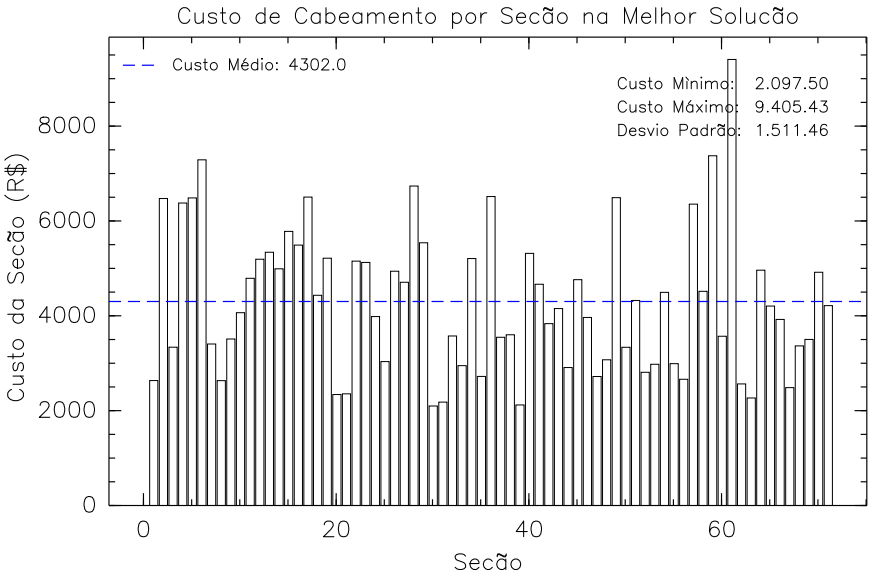


Figura 28: Solução para o Norte da Vila Mariana: custo por seção

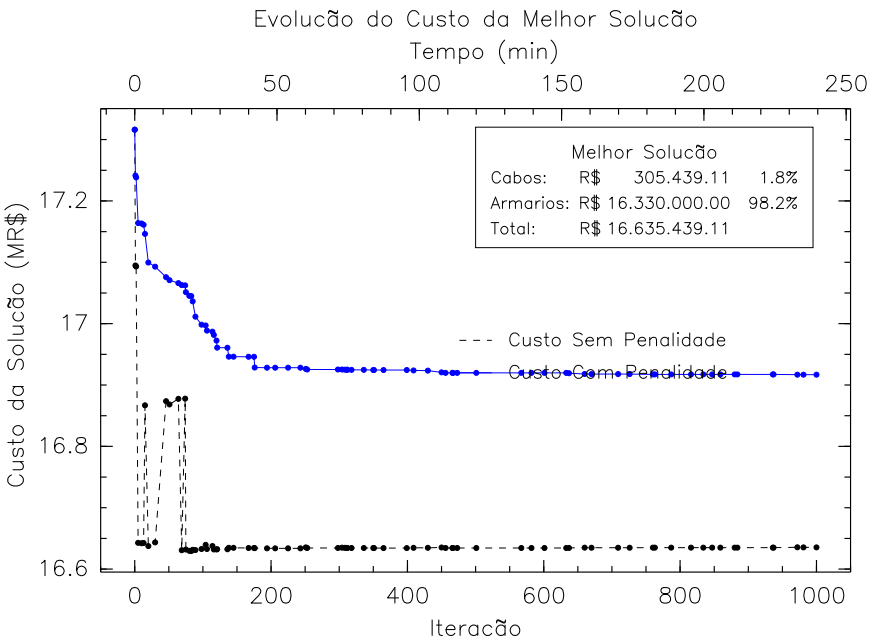


Figura 29: Solução para o Norte da Vila Mariana: custo total por iteração

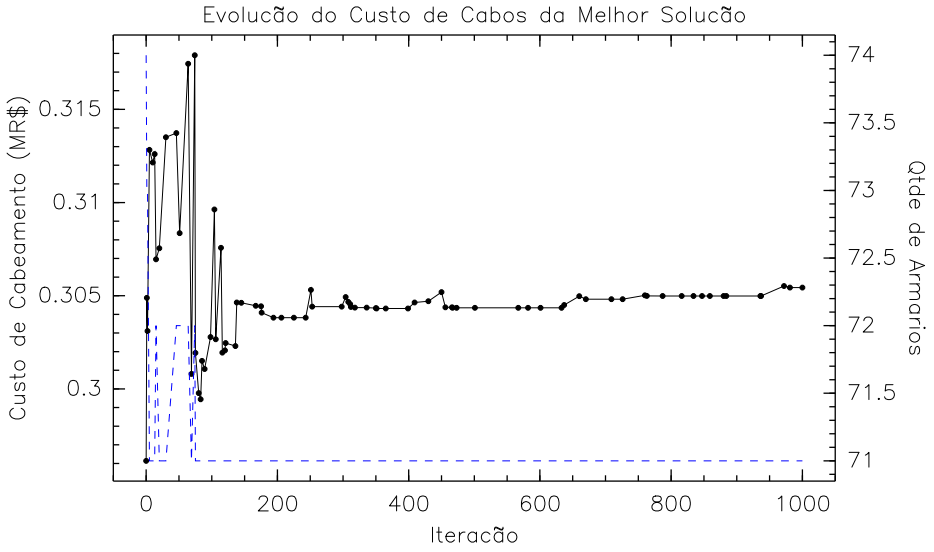


Figura 30: Solução para o Norte da Vila Mariana: custo de cabeamento por iteração

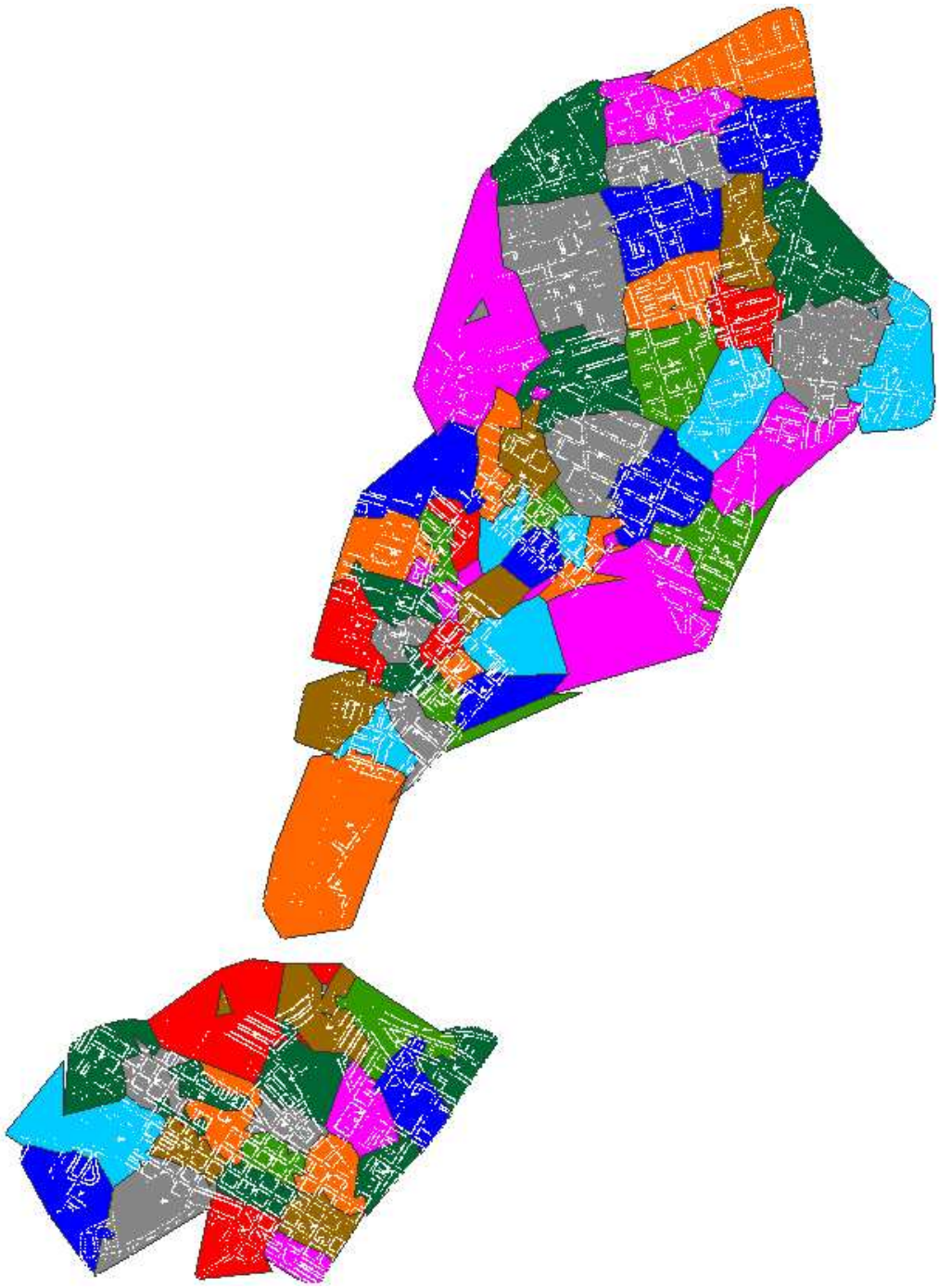


Figura 31: Solução para o Norte da Vila Mariana.

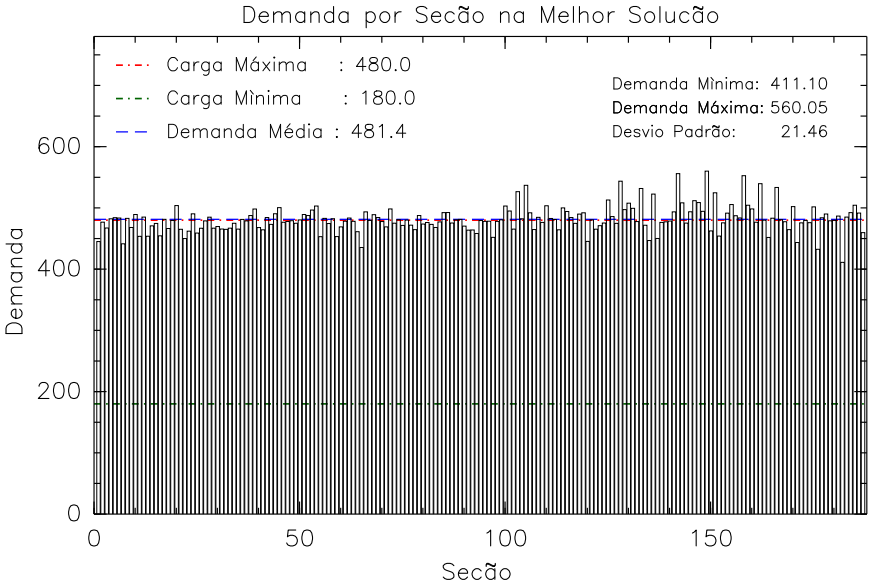


Figura 32: Solução para o do Bairro da Liberdade: demanda por seção.

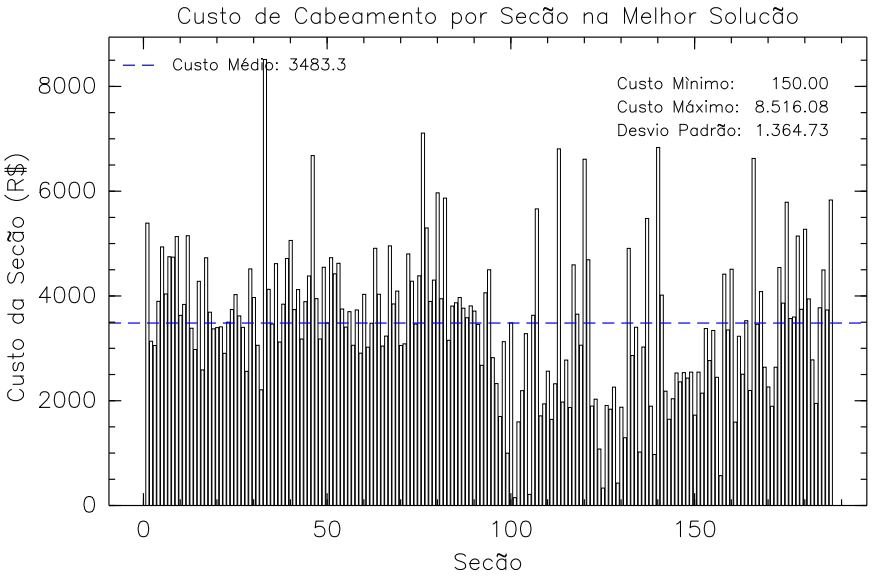


Figura 33: Solução para o do Bairro da Liberdade: custo por seção.

$R\$\ 6.000.000{,}00$ a partir da solução inicial.

Na figura 35 na próxima página representamos a evolução do custo de cabeamento por geração, sem considerar os custos de armários. Por este gráfico conseguimos acompanhar como o algoritmo evolui tentando reduzir o custo de cabeamento. Quando ocorre uma diminuição do número de armários, o que podemos acompanhar pela linha em azul na figura 35, o custo de cabeamento aumenta, até que o algoritmo consegue uma distribuição dos pontos de demanda entre as seções de forma que este custo diminua. Nesta figura, observamos também, como a melhor solução encontrada conseguiu reduzir consideravelmente o número de seções. A solução inicial possuía 213 seções, enquanto a melhor obteve 187, uma diferença de $1.380.000{,}00$ reais.

A figura 36 na página 45 mostra o particionamento dos pontos de demanda em seções de serviço.

5.3.5 Conclusão

A partir dos testes realizados, observamos que a solução final não apresenta formas consideradas ideais geometricamente. Ocorrem alguns pontos dispersos, ou seja, pontos que não estão conexos com a sua seção. Isto se dá pela vizinhança utilizada e está relacionado com as posições vazias da malha de vizinhança. A figura 37 na página 46 ilustra um exemplo de mutação onde ocorre um ponto perdido. Na figura 37(a) temos duas seções, uma vermelha e outra azul. A seção vermelha foi selecionada para aumentar a fronteira, e conforme isso ocorre, ela engloba pontos da seção azul, como ilustrado na figura 37(b), onde o contorno ao redor da seção vermelha indica o quanto a seção aumentou (as posições em branco indicam unidades vazias e não entram para a seção). Se o aumento de fronteira termina na situação ilustrada na figura 37(c) a seção azul fica desconexa, com um ponto disperso.

Nas próximas fases estaremos usando o grafo de arruamento, e poderemos fazer um processamento após a otimização para acertar os pontos perdidos e as fronteiras das seções, de forma a respeitar os limites de quadras e quarteirões.

Referências

[1] S.M. Almeida, S. Livramento, F.K. Miyazawa, A.V. Moura, A.A. dos Santos, and N. Volpato Filho. Algoritmo heurístico para divisão de pontos no plano. Relatório técnico de projeto conjunto cpqd, IC-UNICAMP, 2002.

[2] Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone. *Genetic Programming: An Introduction*. Morgan Kaufmann Publishers, Inc., 1998.

[3] D. A. Goldberg. *Genetic Algorithms in Search. Optimization and Machine Learning*. Addison-Wesley, January 1989.

[4] S. Livramento, S.M. Almeida, F.K. Miyazawa, A.V. Moura, A.A. dos Santos, and N. Volpato Filho. Modelagem em algoritmos genéticos. Relatório técnico de projeto conjunto cpqd, IC-UNICAMP, 2003.

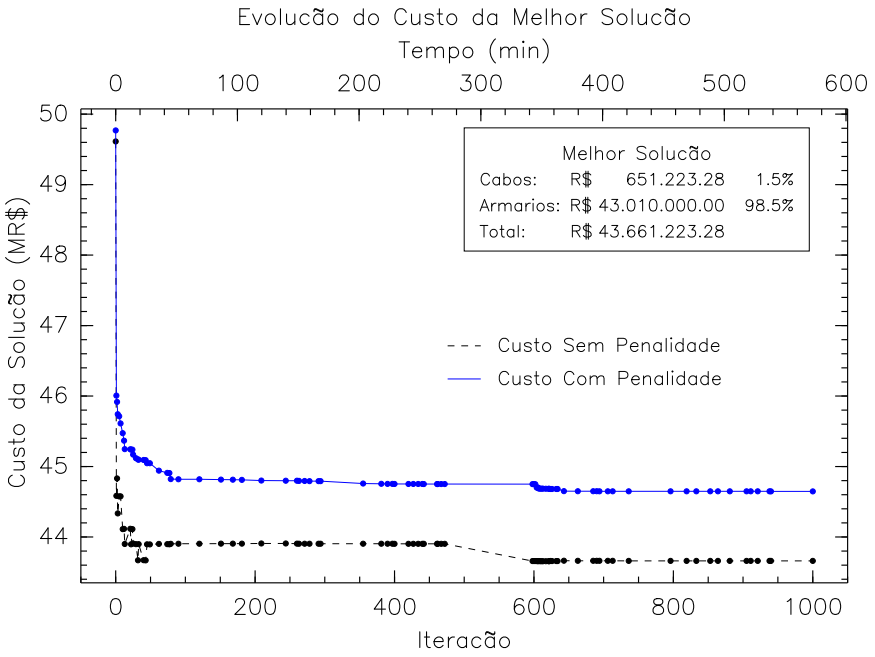


Figura 34: Solução para o do Bairro da Liberdade: custo total por iteração.

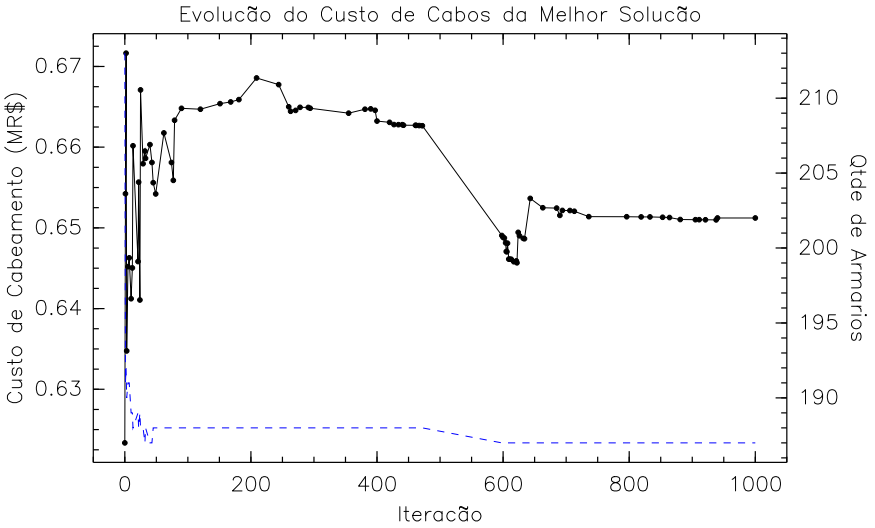


Figura 35: Solução para o do Bairro da Liberdade: demanda por seção.

Figura 36: Solução para o Bairro da Liberdade.

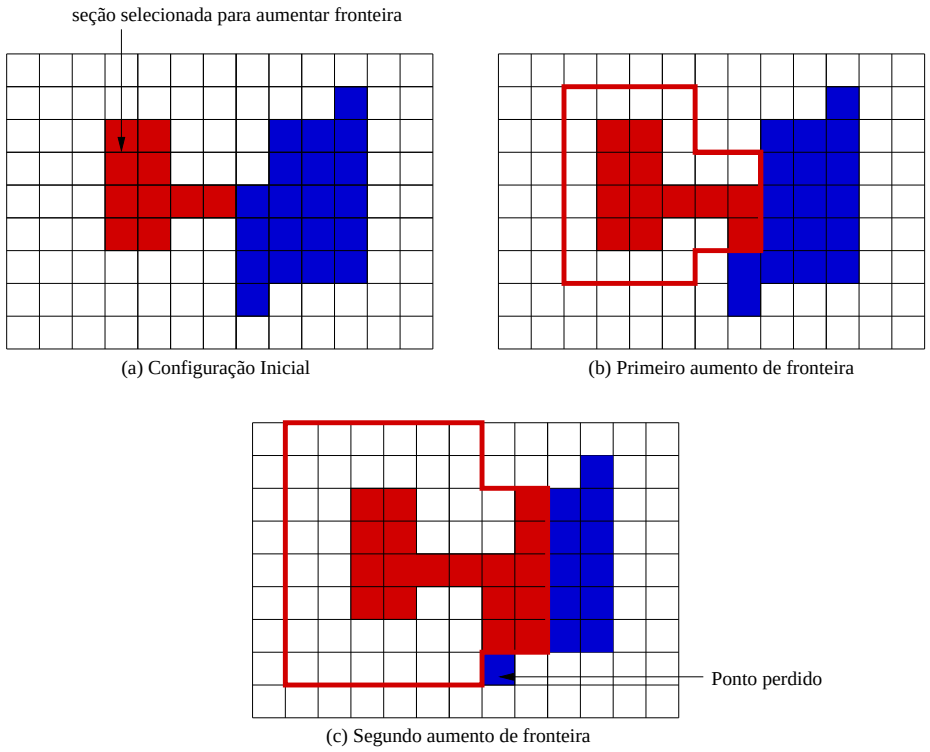


Figura 37: Problema na vizinhança utilizada.

[5] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, third, revised and extended edition edition, 1996.

[6] Zbigniew Michalewicz and David B. Fogel. *How to solve it : modern heuristics*. Springer, 1999.