Unioeste - Universidade Estadual do Oeste do Paraná
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
Colegiado de Ciência da Computação
Curso de Bacharelado em Ciência da Computação

Avaliação do operador *delete-cross* na resolução do Problema do Caixeiro Viajante usando Algoritmos Genéticos

Hendric Gabriel Cechinato

CASCAVEL
2017

HENDRIC GABRIEL CECHINATO

AVALIAÇÃO DO OPERADOR *DELETE-CROSS* NA RESOLUÇÃO DO PROBLEMA DO CAIXEIRO VIAJANTE USANDO ALGORITMOS GENÉTICOS

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação, do Centro de Ciências Exatas e Tecnológicas da Universidade Estadual do Oeste do Paraná - Campus de Cascavel

Orientador: Prof. Dr. Adair Santa Catarina

CASCAVEL
2017

HENDRIC GABRIEL CECHINATO

AVALIAÇÃO DO OPERADOR *DELETE-CROSS* NA RESOLUÇÃO DO PROBLEMA DO CAIXEIRO VIAJANTE USANDO ALGORITMOS GENÉTICOS

Monografia apresentada como requisito parcial para obtenção do Título de Bacharel em Ciência da Computação, pela Universidade Estadual do Oeste do Paraná, Campus de Cascavel, aprovada pela Comissão formada pelos professores:

Prof. Dr. Adair Santa Catarina (Orientador)
Colegiado de Ciência da Computação,
UNIOESTE

Prof. M.Eng. Adriana Postal
Colegiado de Ciência da Computação,
UNIOESTE

Prof. Dr. André Luiz Brun
Colegiado de Ciência da Computação,
UNIOESTE

Cascavel, 1 de dezembro de 2017

DEDICATÓRIA

À todos aqueles que me apoiaram de alguma forma durante esta importante fase da minha vida

AGRADECIMENTOS

Agradeço primeiramente aos meus pais, Joscelito Cechinato e Clarinda Roseli Guerra Cechinato, e minha irmã Carlye Nicheli Cechinato pelos aconselhamentos, incentivos, pelos ensinamentos que contribuíram para a formação de meus valores éticos e morais e também por darem todo o suporte necessário para a conclusão deste longo período de estudos.

Ao meu orientador, Adair Santa Catarina, que ao longo desses três anos de curso me orientou em projetos de pesquisa e contribuiu para que esse momento chegasse.

À minha namorada, Vanessa Suéllen Valgas, por estar sempre presente nos momentos de dificuldades e por pacientemente ceder muitos momentos de diversão para que eu pudesse concluir minhas atividades acadêmicas.

Aos meus tutores do PETComp, Clodis Boscarioli e Marcio Seiji Oyamada, que contribuíram com a minha formação acadêmica através dos projetos de ensino, pesquisa e extensão desenvolvidos ao longo destes três anos de grupo.

E, por fim, à todos os meus professores por terem me ensinado tudo o que eu vou carregar para a minha vida profissional de agora em diante.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

POC	Problema de Otimização Combinatória
PCV	Problema do Caixeiro Viajante
TSP	*Traveling Salesman Problem*
AG	Algoritmo Genético
sTSP	Problema do Caixeiro Viajante Simétrico
aTSP	Problema do Caixeiro Viajante Assimétrico
mTSP	Multi Problema do Caixeiro Viajante
PMX	*Partially Mapped Crossover*
OX	*Ordered Crossover*
CX	*Cyclic Crossover*

Sumário

Resumo

O Problema do Caixeiro Viajante é um problema de pesquisa operacional pertencente à classe dos NP-*Hard*. Seu objetivo é determinar o menor caminho para percorrer uma série de cidades, visitando uma vez cada uma delas, e então retornar à cidade inicial. Estudado em diversas áreas da Ciência da Computação, pode ser solucionado através de heurísticas como os Algoritmos Genéticos, método utilizado neste trabalho. Para obter uma melhor qualidade nas soluções, novos métodos podem ser incorporados aos Algoritmos Genéticos. Neste trabalho foi empregado o operador *delete-cross*, um método heurístico responsável por eliminar arestas que se cruzam no roteiro do caixeiro viajante que acelera a convergência mas possui alta complexidade de tempo. Nesse contexto, foi proposto uma variação nomeada de 1-*delete-cross* que elimina apenas um cruzamento de arestas a cada geração evoluída. Por fim, três abordagens foram implementadas e testadas: um Algoritmo Genético genérico, um Algoritmo Genético com o 1-*delete-cross* substituindo o método de mutação e um Algoritmo Genético com a inclusão do operador *delete-cross*. O objetivo dos testes foi avaliar a qualidade das soluções e o número de gerações produzidas para se obter os resultados. Os testes realizados mostraram que o operador *delete-cross* proporciona soluções muito melhores que os outros métodos avaliados. Entretanto, o tempo de processamento chega a ser doze vezes mais lento que as outras abordagens. O operador 1-*delete-cross* proporciona resultados melhores que o AG clássico com pouca diferença em relação ao tempo de convergência.

Palavras-chave: Problema do Caixeiro Viajante, Algoritmos Genéticos, Operadores Genéticos, *delete-cross*.

Capítulo 1

Introdução

Os problemas de otimização combinatória (POC), na sua forma geral, tem como objetivo maximizar ou minimizar uma função definida sobre certo domínio finito [1]. Em um POC temos uma função objetivo e um conjunto de restrições, ambos relacionados às variáveis de decisão. Os valores possíveis das variáveis de decisão são delimitados pelas restrições impostas sobre essas variáveis, formando um conjunto discreto de soluções factíveis a um problema. A resposta para um problema de otimização, chamada de ótimo global, será o menor (ou maior) valor possível para a função objetivo para qual o valor atribuído não viole nenhuma restrição [2].

O Problema do Caixeiro Viajante (PCV) ou *Traveling Salesman Problem* (TSP), objeto de estudo deste trabalho, é um POC que possui inúmeras aplicações práticas, como por exemplo: roteamento de veículos, cabeamento de computadores e perfuração de placas de circuitos integrados [3].

O objetivo do PCV, em casos de roteamento, consiste em percorrer todas as cidades em um percurso e voltar à cidade inicial de forma que a soma dos custos dos deslocamentos seja mínima. Existem diversos métodos de otimização para solucionar o PCV, tais como Subida de Encosta [4], Pesquisa Tabu [5], Arrefecimento Simulado [6], Enxame de Partículas [7], Colônia de Formigas [8] e Algoritmos Genéticos [9]. Neste trabalho, foi experimentado o operador *delete-cross* [10] em um algoritmo genético para resolver o PCV.

Algoritmos Genéticos (AGs) são uma meta-heurística inspirada pelo processo de seleção natural. Sua fundamentação teórica foi a tese de doutorado apresentada por John Henry Holland e publicada em livro no ano de 1975 [11]. Seu método consiste em mover uma população de cromossomos, para uma nova população, usando princípios de seleção natural juntamente com

operadores inspirados pela genética, tais como recombinação, mutação e inversão. É objetivo dos AGs fazer a população convergir em um valor ótimo de adaptação [12]. Informações mais detalhadas sobre os AGs serão apresentadas na Seção 2.1.

Em função do problema a ser resolvido, os operadores, em especial o operador de recombinação, podem ser codificados em diferentes algoritmos, assim como novos operadores podem ser propostos. Um exemplo de operador proposto, especificamente para o PCV, é o *delete-cross*, que consiste em excluir os cruzamentos entre arestas de um percurso para acelerar o processo de convergência do AG.

Neste trabalho, foi empregado o operador *delete-cross* em um AG, combinando-o com os operadores de recombinação e mutação. Diferentes estratégias de aplicação do operador *delete-cross* foram propostas e analisadas. Usando estas estratégias, foram realizados testes para verificar o funcionamento do operador em relação à qualidade das soluções encontradas.

Este trabalho justifica-se pela ausência de um trabalho que explore o operador *delete-cross* quando associado a um AG em diferentes estratégias, na perspectiva de obter bons resultados em diferentes instâncias do PCV, minimizando o número de gerações necessárias para a convergência.

Este trabalho está organizado da seguinte forma:

- **Capítulo 2:** apresenta os conceitos de Algoritmos Genéticos e o Problema do Caixeiro Viajante, bem como a adaptação de um AG para resolvê-lo.
- **Capítulo 3:** exibe o AG implementado, as estratégias *delete-cross* propostas, seus métodos e seu funcionamento.
- **Capítulo 4:** expõe os resultados obtidos nas diferentes abordagens e discussões.
- **Capítulo 5:** apresenta as sugestões para trabalhos futuros no âmbito deste.

Capítulo 2

Fundamentação Teórica

Neste capítulo serão apresentados e explicados os conceitos necessários sobre o Problema do Caixeiro Viajante e Algoritmos Genéticos. Ademais, é mostrado como um AG é adaptado para solucionar o PCV. Partindo dos princípios descritos na introdução deste trabalho, é apresentado como os operadores de seleção, recombinação e mutação são aplicados ao PCV de forma que sejam gerados filhos viáveis no processo de evolução. Por fim, o operador *delete-cross* é apresentado de forma mais detalhada, bem como os trabalhos em que foi utilizado e suas diferentes abordagens.

2.1 O Problema do Caixeiro Viajante

O Problema do Caixeiro Viajante é um dos problemas mais conhecidos de otimização combinatória. Ele pode ser descrito como: dado um grafo completo $G = (V, A)$ com n vértices, seu objetivo é encontrar um ciclo hamiltoniano de custo mínimo. Ou seja, deseja-se, a partir de um vértice inicial, passar por todos os demais vértices do grafo apenas uma vez e voltar para o vértice inicial. Cada aresta que liga um vértice i a j no grafo possui um custo fixo (c_i, c_j) que determina quanto se gasta no deslocamento de i para j. Este custo pode ter diferentes significados de acordo com a aplicação desejada. A solução final é dada pela soma dos pesos das arestas selecionadas do grafo G [13]. A Figura 2.1 exemplifica uma solução para o jogo de Hamilton.

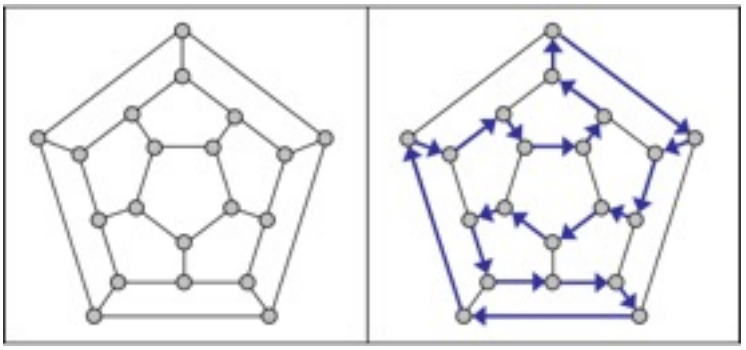

Figura 2.1: Exemplo de solução para o jogo de Hamilton
Fonte: Ganhoto [14].

Em casos de roteamento, o conjunto de n cidades conhecidas $C = \{C_1, C_2, ..., C_n\}$ correspondem aos vértices do grafo, e a distância $d(c_i, c_j)$ entre duas cidades aleatórias estão associadas às arestas do grafo. Neste caso, a solução do problema consiste em encontrar o percurso de menor distância que conecta todas as cidades, visitando-as uma única vez.

A definição formal do PCV pode ser escrita como [13]:

Dado um grafo completo $G(V, A)$ e uma função custo $c : V$ x $V \rightarrow \mathbb{Q}^+$, encontrar uma permutação $(v_1, v_2, ..., v_n)$ de V tal que

$$x = c(v_n, v_1) + \sum_{i=1}^{n-1} c(v_i, v_{i+1}) \tag{2.1}$$

seja mínimo.

Além disso, o Problema do Caixeiro Viajante pode ser classificado como PCV simétrico (sTSP), PCV assimétrico (aTSP) e multi PCV (mTSP). Tais classificações são descritas em Matai *et al.* [3] como:

- **sTSP:** é como a definição formal descrita anteriormente. Neste caso, o custo de deslocamento da cidade i até j é o mesmo que de j até i, ou seja, $d(i, j) = d(j, i)$.

- **aTSP:** se $d(i, j) \neq d(j, i)$ ocorre em pelo menos um caso de (i, j), então o PCV torna-se assimétrico.

- **mTSP:** consiste em encontrar rotas para n caixeiros viajantes que deixam um único armazém. Existem variações para o mTSP, em que são tratadas abordagens com um único

ou múltiplos armazéns, número fixo ou variável de caixeiros viajantes ou custo de deslocamento variável quando o número de caixeiros viajantes também é variável.

2.2 Algoritmos Genéticos

Algoritmos Genéticos são inspirados na maneira como o darwinismo explica o processo de evolução das espécies. Através dos princípios da evolução e da seleção natural, podem ser aplicados em uma série de problemas, especialmente os de otimização [12].

Em AGs, o termo **cromossomo** tipicamente se refere a um candidato para a solução de um problema. O cromossomo consiste em um conjunto de **genes** que codificam um elemento particular de um cromossomo. Cada instância particular de um bit em um gene é chamada de **alelo** e a posição em que o mesmo se encontra é denotada **locus**. Associado a cada cromossomo há um valor que expressa a aptidão do mesmo, normalmente chamado de ***fitness***. Este valor define quão bem o cromossomo resolve um problema [15]. Neste trabalho, os termos cromossomo e indivíduo são utilizados sem distinção.

O funcionamento de um AG clássico consiste, primeiramente, na geração de uma **população** inicial de indivíduos, que é formado por um conjunto aleatório de cromossomos. Então, dá-se início ao processo evolutivo onde cada cromossomo é avaliado e classificado através de seu *fitness*. Após a etapa de avaliação ocorre a fase de seleção.

Em geral, os cromossomos mais aptos são selecionados e os menos aptos são descartados. Os membros selecionados podem sofrer modificações em suas características fundamentais através dos operadores genéticos, gerando descendentes para a seguinte geração. Na etapa de substituição dos cromossomos inaptos para a próxima população, pode ocorrer o **elitismo**. Quando isto ocorre, a melhor solução da população i é copiada para a população $i + 1$ [16]. A Seção 2.2.1 descreve mais detalhadamente as características dos operadores genéticos. O processo de evolução é repetido até que uma solução satisfatória seja encontrada.

2.2.1 Operadores Genéticos

A forma mais simples de um AG envolve três tipos de operadores, os quais são descritos como [15]:

- **Seleção:** consiste em selecionar, probabilisticamente, os cromossomos que se reproduzi-

rão em uma população. Quanto maior for seu grau de aptidão, maior será a probabilidade do mesmo se reproduzir.

- **Recombinação:** escolhe, aleatoriamente, um ou mais locus de um cromossomo e realiza a troca entre as subsequências de alelos anterior e próxima ao locus, gerando um novo cromossomo. A Figura 2.2 ilustra o processo de recombinação de um ponto. O corte ocorre entre as posições 4 e 5 dos indivíduos pais p_1 e p_2. A reprodução gera os filhos f_1 e f_2.

- **Mutação:** inverte, aleatoriamente, um ou mais alelos de um cromossomo. Por exemplo, o indivíduo representado por 10011100 pode sofrer mutação em sua quarta posição, transformando-o no indivíduo 10001100.

  O operador de **recombinação** também pode ser chamado de operador de **cruzamento**. Entretanto, o termo cruzamento foi utilizado neste trabalho para se referir ao cruzamento entre arestas de um percurso do caixeiro viajante.

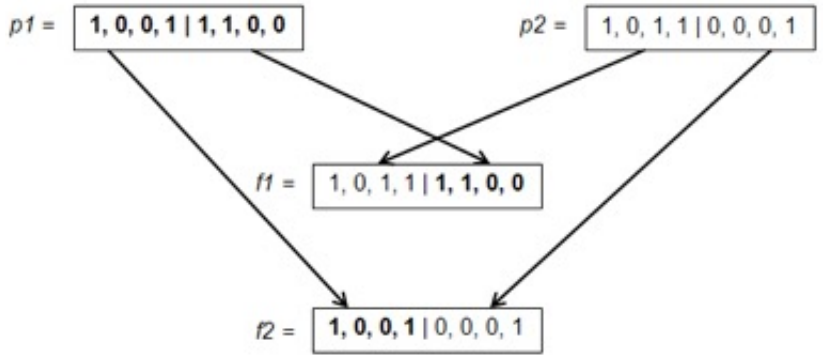

Figura 2.2: Exemplo de recombinação de um ponto

Além disso, há um quarto operador proposto por Holland chamado de inversão, descrito por Mitchell [15]:

- **Inversão:** consiste em escolher duas posições de um cromossomo e as inverter. Por exemplo, utilizando o operador de inversão no indivíduo 11101000 nos pontos 1 e 4, transformando-o no indivíduo 01111000.

Sendo assim, é possível resumir o funcionamento de um AG genérico através do Algoritmo 1 [15]:

Algoritmo 1: AG TÍPICO

gerar uma população aleatória de cromossomos (candidatos a solução do problema)
calcular o *fitness* de cada indivíduo na população
repita
  selecionar um par de cromossomos da população atual, através do operador de seleção
  através do operador de recombinação, realizar o cruzamento dos pais para gerar dois novos filhos
  realizar a mutação dos novos filhos, através da inversão de um gene aleatório do cromossomo
  avaliar o *fitness* dos novos indivíduos gerados
  substituir os indivíduos considerados inaptos na população
até *que o critério de parada seja atendido*;

2.3 Algoritmo Genético aplicado ao Problema do Caixeiro Viajante

Os algoritmos genéticos podem ser adaptados para resolver diversos tipos de problemas. No caso do PCV, as adequações envolvem a codificação de uma rota no cromossomo e operadores específicos de recombinação e mutação. As próximas seções apresentam como um AG pode ser adaptado para resolver o PCV.

2.3.1 Representação Cromossômica

Conforme mencionado em Benevides [17], a representação cromossômica mais simples do espaço de busca para o PCV é a representação por caminho. Ou seja: cada cidade do circuito é numerada e cada posição do cromossomo recebe o número que identifica uma cidade. A rota é representada pelo próprio cromossomo, por isso ambos os termos são tratados como sinônimos nesta seção. Exemplificando, o cromossomo $(1, 3, 2, 4, 5, 6, 7)$ caracteriza a rota demonstrada na Figura 2.3.

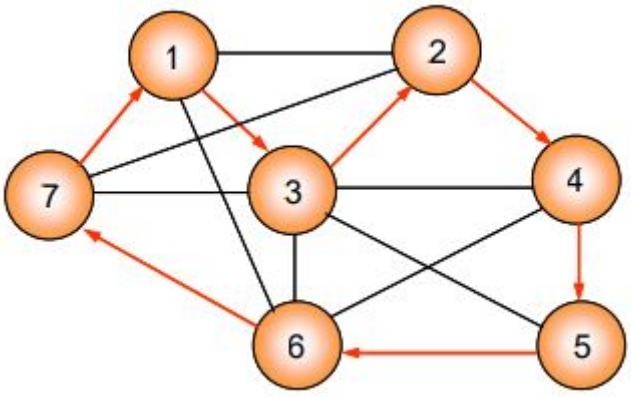

Figura 2.3: Grafo de exemplo para o PCV
Fonte: Benevides [17].

Em um AG aplicado ao PCV, o termo **cromossomo** está relacionado à rota do caixeiro viajante e o termo **alelo** se refere a uma cidade presente no roteiro. Já o termo **população** consiste no conjunto de rotas que são tratadas como possíveis soluções para o problema e são evoluídas durante o processo de reprodução.

2.3.2 Operador de Seleção

A etapa de seleção em um AG aplicado ao PCV consiste em escolher, probabilisticamente, duas rotas com o objetivo de aplicar os métodos de reprodução para a geração de novas rotas candidatas à solução ótima.

Existem vários métodos para a seleção dos cromossomos pais, alguns descritos por Lucas e Álvares [12] como:

- **Seleção por *ranking*:** os indivíduos da população são ordenados de acordo com o seu *fitness* e sua probabilidade de escolha é atribuída conforme a posição que ocupam.
- **Seleção por giro de roleta:** calcula-se o somatório do *fitness* (total) da população e sorteia-se um valor i que corresponda ao intervalo $[0; total]$. Seleciona-se o indivíduo que corresponda à faixa do somatório onde i se localiza.
- **Seleção por torneio:** grupos de soluções são escolhidos sucessivamente e as que apresentarem melhor *fitness* são selecionadas.

- **Seleção uniforme:** todos os indivíduos possuem a mesma probabilidade de serem selecionados. Entretanto, esta forma de solução pode acarretar uma piora da população sobre qual atua.

As Figuras 2.4 e 2.5 ilustram os métodos de seleção descritos nesta seção. Na Figura 2.4 ocorre o processo de seleção entre cinco rotas. Na Figura 2.4 (A), as rotas que possuem melhor *fitness* ocupam uma probabilidade sucintamente maior de serem escolhidas. Na Figura 2.4 (B), a probabilidade é distribuída de acordo com o *fitness* de cada rota. Na Figura 2.4 (C), todas as rotas possuem a mesma probabilidade de serem escolhidas.

Figura 2.4: Probabilidades de escolhas nos métodos de seleção por *ranking* (A), giro da roleta (B) e uniforme (C)

Na Figura 2.5 há um exemplo do processo de seleção por torneio. Nesse caso, dois pares de rotas são selecionados e avaliados de acordo com seus *fitness* (A). Em um problema de maximização do *fitness*, os maiores de cada torneio são escolhidos e avançam para um novo torneio (B) até que reste apenas um, que é o selecionado (C).

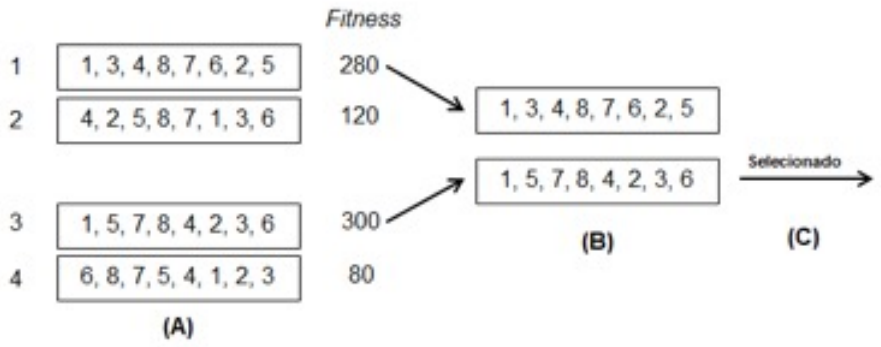

Figura 2.5: Processo de seleção por torneio

2.3.3 Operador de Recombinação

Conforme apresentado na Seção 2.2.1, o operador de recombinação cruza as características dos cromossomos pais, gerando novos indivíduos. Entretanto, métodos convencionais de recombinação, como o em um ponto, podem gerar filhos inviáveis. A Figura 2.6 mostra a recombinação entre as rotas $p_1 = (1, 2, 4, 8, 6, 5, 7, 3)$ e $p_2 = (3, 5, 8, 6, 1, 2, 4, 7)$. É possível notar que os filhos gerados não representam circuitos hamiltonianos, pois apresentam repetições de cidades em suas rotas, além de deixar outras de fora do roteamento. O filho f_1, por exemplo, representado por $(3, 5, 8, 6, 6, 5, 7, 3)$, possui repetições das cidades 3, 5 e 6, deixando de fora as cidades 1, 2 e 4.

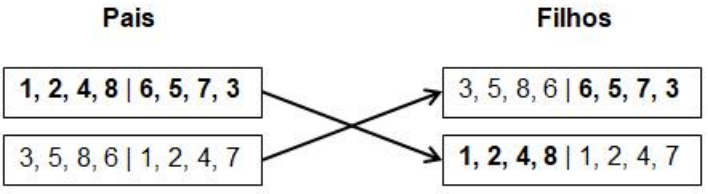

Figura 2.6: Recombinação de um ponto entre dois cromossomos que representam rotas do PCV

Nestes casos é necessário que sejam aplicados operadores de recombinação que garantam que ocorra apenas uma instância de cada cidade do circuito. Apresentados por Goldberg e Lingle [18], Oliver *et al.* [19] e Davis [20], os operadores PMX, CX e OX, respectivamente, são operadores específicos para o PCV, descritos por Benevides [17] como:

- ***Partially Mapped Crossover* (PMX):** faz uma recombinação parcialmente mapeada utilizando dois pontos de corte e fazendo a recombinação normalmente entre os dois pontos. O processo consiste na troca entre um vértice do primeiro pai por um vértice do segundo pai para cada posição da rota entre os pontos, onde cada uma destas trocas define um mapeamento. A Figura 2.7 ilustra um exemplo do processo de recombinação através do operador PMX.

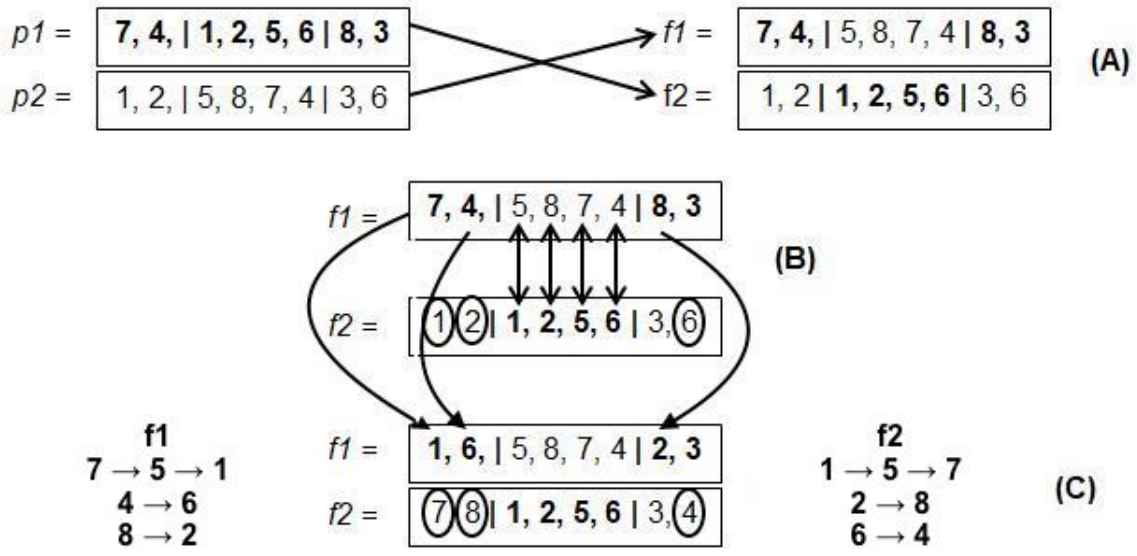

Figura 2.7: Exemplo de recombinação usando o operador PMX

Na figura 2.7 (A) é possível observar que o filho f_1, por exemplo, recebe, antes e após os pontos de corte, as cidades de p_1 e, entre os pontos de corte, as cidades de p_2. Porém, esse processo frequentemente gera filhos inviáveis, como f_1 que repetiu as cidades 7, 4 e 8. Para resolver esse problema é feito o mapeamento entre os cromossomos pais para cada cidade entre os pontos de corte, como demonstrado nas setas verticais em (B), $(5 \leftrightarrow 1, 8 \leftrightarrow 2, 7 \leftrightarrow 5, 4 \leftrightarrow 6)$. Por fim, é necessário substituir transitivamente as cidades repetidas pelas mapeadas. O mapeamento transitivo pode ocorrer quando uma cidade mapeia outra que já está mapeando uma outra cidade, como mostrado em f_1 (C), $7 \rightarrow 5 \rightarrow 1$, por exemplo. Ao substituir as cidades repetidas pelas mapeadas, como mostrado através das setas curvadas para f_1 e pelos círculos em f_2, são gerados filhos viáveis.

- ***Ordered Crossover* (OX):** é um operador de recombinação de dois pontos de corte onde os filhos gerados herdam a ordem relativa da sequência de cidades dos pais. A Figura 2.8 ilustra um exemplo do processo de recombinação através do operador OX.

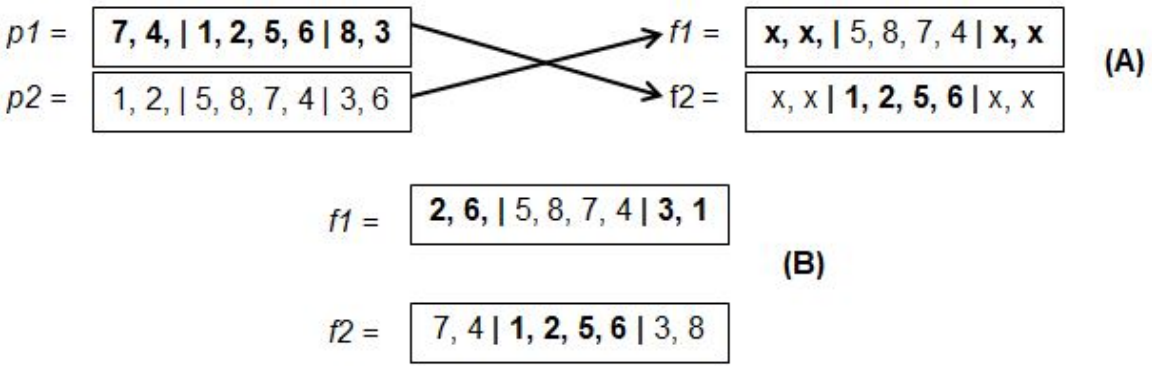

Figura 2.8: Exemplo de recombinação usando o operador OX

Na Figura 2.8 (A) é possível observar que os filhos herdam as cidades que estão entre os pontos de corte de cada pai. Após essa etapa, constrói-se uma lista que inicia a partir do último ponto de corte do pai p_2 $(3, 6, 1, 2, 5, 8, 7, 4)$. Removendo as cidades que já estão no roteiro do filho f_2, é obtida a sequência 3, 8, 7, 4. A sequência 3, 8 é inserida após o segundo ponto de corte de f_2 e a sequência 7, 4 antes do primeiro ponto de corte de f_2. De forma análoga, da lista do pai p_1 $(8, 3, 7, 4, 1, 2, 5, 6)$ são removidos 5, 8, 7, 4, gerando a sequência 3, 1, 2, 6 que são inseridas em f_1 da mesma maneira que em f_2.

- ***Cyclic Crossover* (CX):** é um operador que não utiliza pontos de corte. Ele executa a recombinação de modo que cada gene das descendências vem das posições correspondentes de qualquer um dos pais. A Figura 2.9 ilustra o processo de recombinação através do operador CX.

  O primeiro passo do operador de recombinação CX é fazer a cópia da primeira cidade de p_1 para f_1, como exposto na Figura 2.9 (A). Em seguida, observa-se qual elemento em p_2 corresponde à mesma posição de p_1, que em (B), é 4. Então é verificado em p_1 a posição onde se encontra a cidade 4 (no caso, terceira posição) e replica-se em f_1 o elemento na posição encontrada (cidade 4 na terceira posição), como visto em (C).

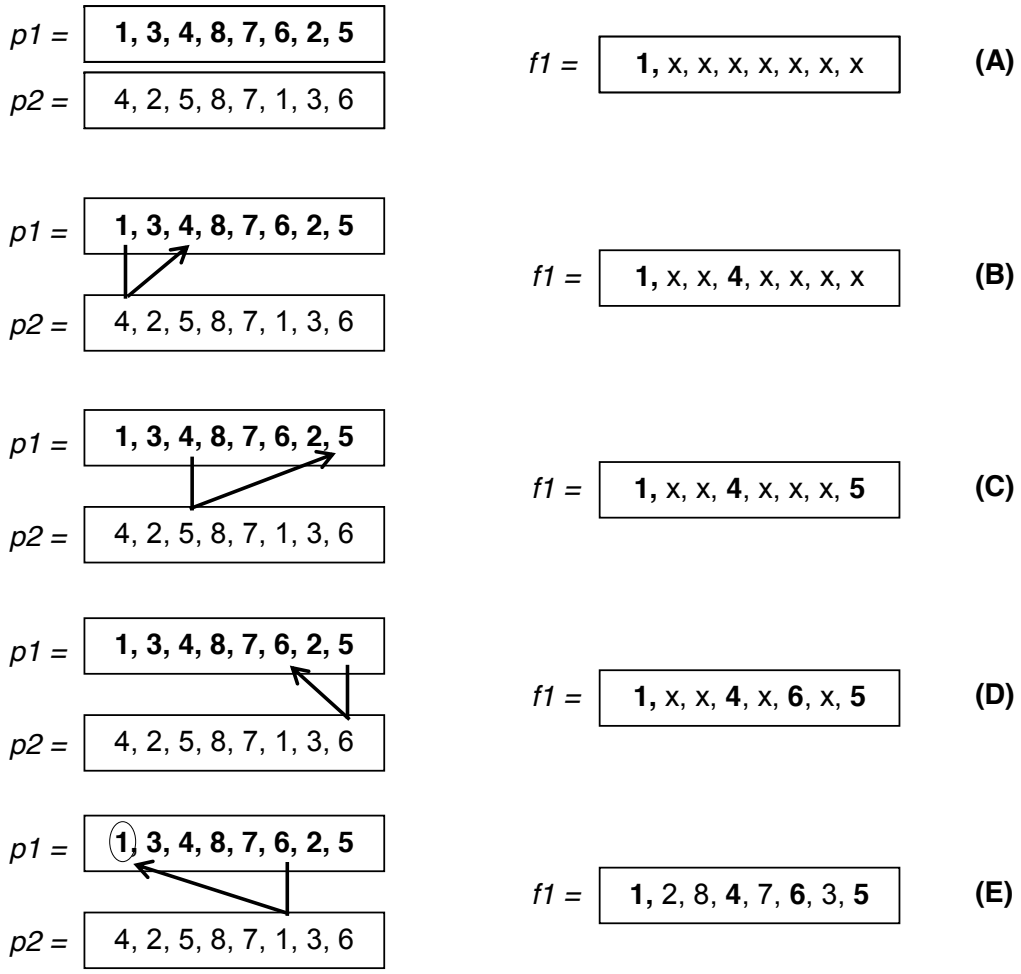

Figura 2.9: Exemplo de recombinação usando o operador CX

O processo é repetido até que o elemento de p_2, a ser inserido, corresponda a um elemento de p_1 já herdado por f_1, como ocorre em (E) com o elemento 1. Quando isso ocorre, os elementos faltantes são inseridos na ordem que se apresentam em p_2, portanto, $(2, 8, 7, 3)$ nas posições 2, 4, 5 e 7.

2.3.4 Operador de Mutação

Como descrito na Seção 2.2.1, o operador de mutação consiste em alterar aleatoriamente o valor de um dos alelos do cromossomo. No caso do PCV, é necessário que seja utilizado o tipo de mutação por troca, que consiste na troca de posição entre dois genes [17]. Ou seja, caso ocorra mutação de um cromossomo, duas cidades da rota são selecionadas aleatoriamente

e suas posições são trocadas. A Figura 2.10 ilustra este processo.

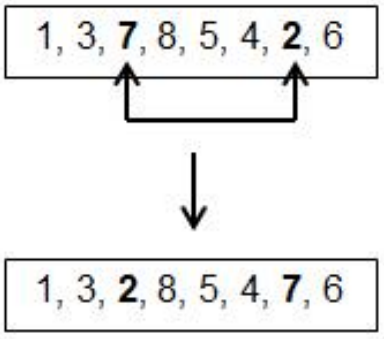

Figura 2.10: Exemplo do operador mutação aplicado ao PCV

A probabilidade de se efetuar uma mutação deve ser relativamente baixa, caso contrário o algoritmo se comportará realizando uma busca aleatória, dificultando a convergência [17].

2.4 O Operador *delete-cross*

A heurística 2-opt, apresentada por Croes [21] e mencionada por Shi *et al.* [10] com o nome de *delete-cross*, é um processo que pode ser adicionado após as etapas de recombinação e mutação para remover os cruzamentos que existem entre as arestas do percurso, conforme apresentado na Figura 2.11 [10].

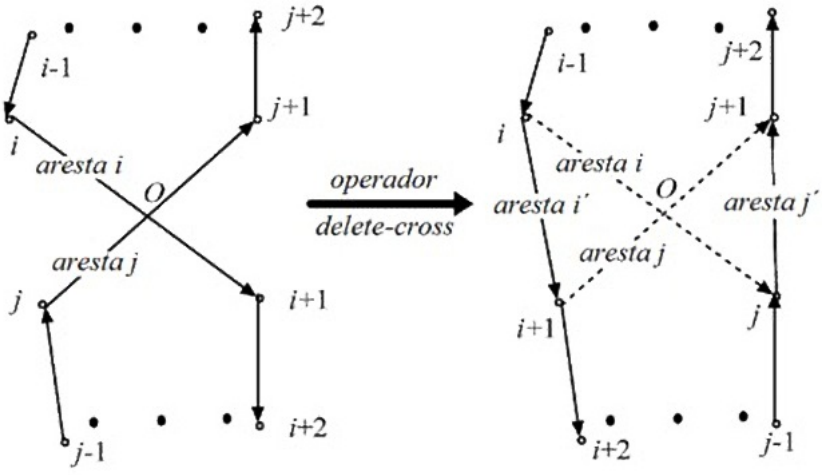

Figura 2.11: O operador *delete-cross*.
Fonte: Adaptado de Shi *et al* [10]

Ao avaliar que duas arestas do percurso se cruzam, o operador refaz o caminho invertendo extremos destas arestas. Na Figura 2.11 percebe-se que a aresta i (vértice i a $i+1$) cruza a aresta j (vértice j a $j+1$). Neste caso, o *delete-cross* refaz o percurso invertendo os vértices extremos das arestas i e j.

O método acelera a convergência do AG pois, ao eliminar cruzamentos entre arestas, reduz-se a distância do percurso total. De acordo com a teoria, a soma dos comprimentos de dois lados de um triângulo é maior que o terceiro lado, conforme a inequação 2.2 [10].

$$d(i, O) + d(i+1, O) > d(i, i+1) \tag{2.2}$$

Existem trabalhos em diferentes abordagens que utilizaram o operador em algoritmos que tentam solucionar o PCV. Em Shi *et al.* [10] foi utilizado na otimização por enxame de partículas. Em Pan *et al.* [22] o operador foi utilizado em um algoritmo de imunidade juntamente com um algoritmo guloso. Baseado nesses três algoritmos foi criado um algoritmo de imunidade híbrido para resolver o PCV. Em Wu e Ouyang [23], foi utilizado em um algoritmo de colônia de formigas híbrido.

Para explicar como a heurística é utilizada em um AG aplicado ao PCV, o Algoritmo 2 detalha o funcionamento do *delete-cross* [10]:

Algoritmo 2: OPERADOR *delete-cross*

```
1 para cada cidade i até i − 3 no roteiro faça
2     para cada cidade j no roteiro faça
3         se cruzamento(aresta[i], aresta[j]) == verdadeiro) então
4             para cada cidade k até (j - i)/2 faça
5                 troque(j − k, i + k + 1)
6             fim
7         fim
8     fim
9 fim
```

O algoritmo consiste em avaliar vértice por vértice se ocorre um cruzamento entre as arestas $\overline{i, i+1}$ e $\overline{j, j+1}$. Quando a interseção é detectada, como na linha 3, todas as cidades pertencentes ao roteiro do vértice i ao j são invertidas até a metade deste caminho, removendo o cruzamento entre as arestas.

O operador *delete-cross* não garante que todos os cruzamentos serão removidos na primeira

evolução, pois é possível que a remoção de duas arestas cruzadas venha a criar um novo cruzamento com uma aresta que possui um vértice que já foi avaliado na mesma execução. À medida que os cruzamentos são reduzidos, o algoritmo tende a criar uma rota sem que nenhum conjunto de arestas se cruzem.

Capítulo 3

Implementação de um AG Clássico e Diferentes Abordagens *delete-cross* para o PCV

O PCV é um dos problemas de otimização clássicos estudado por diversos autores. Para comparar diferentes algoritmos e estratégias para resolução desta classe de problemas, Reinelt [24] construiu uma base de testes, chamada TSPLIB. A seção seguinte apresentará mais informações sobre a biblioteca. Na sequência será apresentada a implementação de um AG clássico para resolver o PCV. Em seguida, serão apresentadas as diferentes abordagens *delete-cross* incorporadas ao Algoritmo Genético clássico para acelerar a sua convergência.

Os resultados obtidos com a abordagem clássica foram utilizados, em combinação com os resultados ótimos registrados na TSPLIB, para avaliar a eficiência e eficácia de um AG que empregue o operador *delete-cross*, em diferentes abordagens.

3.1 Especificação dos Problemas

Os problemas utilizados para a realização dos testes são provenientes da TSPLIB [24], que é uma biblioteca que possui casos de teste de várias instâncias para o PCV. As entradas de dados estão em arquivos de texto simples nomeados por uma palavra e um número, que geralmente representam a localização e o número de cidades presentes no caso. Nestes arquivos, a representação das localizações das cidades pode ser apresentada como uma matriz de adjacência ou uma lista de coordenadas de cada vértice. Neste trabalho foram utilizados apenas arquivos cuja representação é a lista de coordenadas.

A formatação da entrada, na representação escolhida, para cada vértice pertencente ao grafo do PCV é dada por três valores numéricos, em que:

- O primeiro valor corresponde a um número sequencial que identifica a cidade;
- O segundo valor corresponde à coordenada x da cidade;
- O terceiro valor corresponde à coordenada y da cidade.

Para a realização dos testes, foram escolhidos apenas casos simétricos (sTSP) da biblioteca, os quais são mostrados através da Tabela 3.1.

Tabela 3.1: Casos utilizados da TSPLIB

Arquivo	Nº de Cidades	Solução Ótima
eil51	51	426
kroA100	100	21282
a280	280	2579
rd400	400	15281
d657	657	48912

3.2 Algoritmo Genético Clássico

Com o objetivo de realizar os testes para obter os resultados da execução de um AG clássico na resolução do PCV, implementou-se, em linguagem de programação Java, um algoritmo que recebe como entrada a lista de coordenadas das cidades do circuito hamiltoniano e cria uma população aleatória de rotas válidas para aplicar os métodos de evolução do algoritmo. Os resultados obtidos com este AG foram usados como referência para comparação com os algoritmos implementados neste trabalho que utilizam as estratégias *delete-cross* apresentadas na Seção 3.3. A Figura 3.1 apresenta o diagrama de classes do AG clássico implementado.

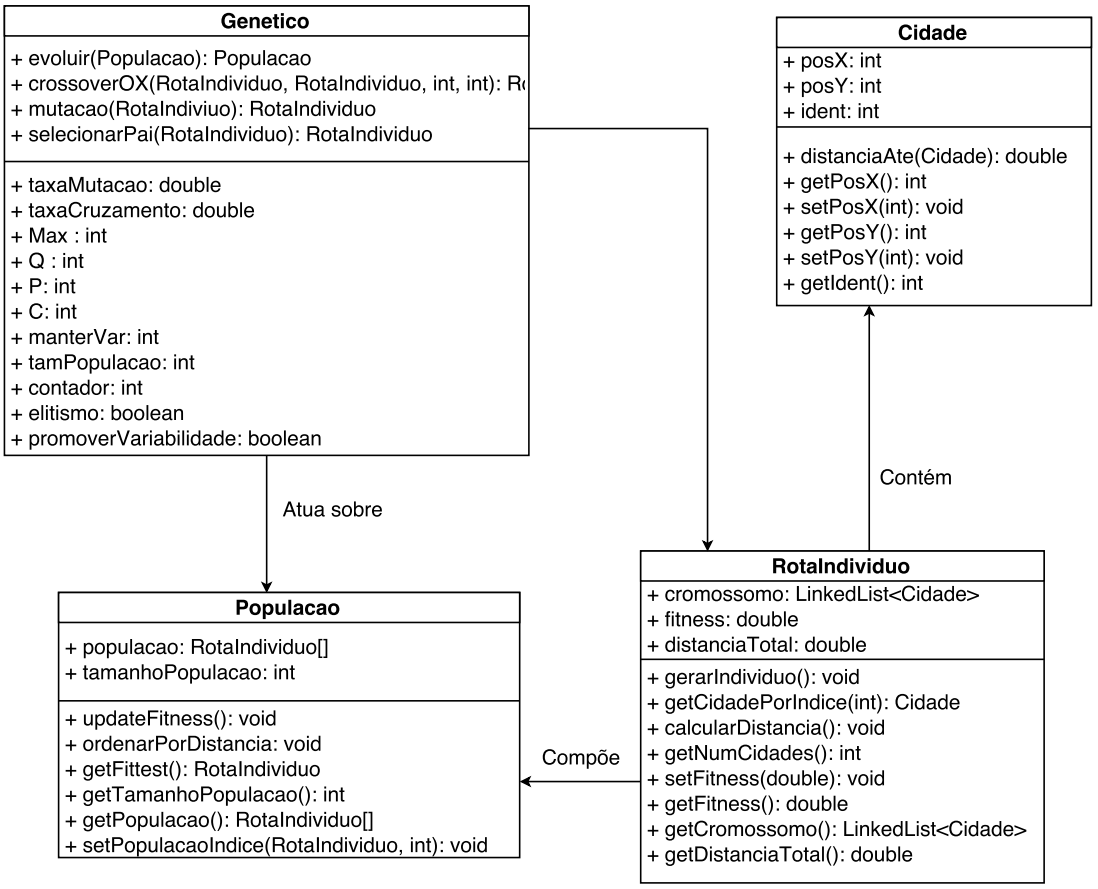

Figura 3.1: Diagrama de classes do AG clássico

As classes pertencentes a essa estrutura implementam os métodos apresentados na Seção 2.3. Elas podem ser descritas como:

- **Cidade:** armazena as coordenadas e o número identificador de cada cidade da entrada de dados. Possui um método que calcula a distância euclidiana entre duas cidades.
- **RotaIndivíduo:** representa um cromossomo. Armazena uma lista de cidades, seu *fitness* correspondente e a distância total do circuito. Possui, principalmente, os métodos de criação dos indivíduos e de cálculo da distância total do roteiro.
- **População:** representa uma população de cromossomos, ou seja, possui uma lista de rotas candidatas a soluções. Possui, principalmente, os métodos de atualização do *fitness* de cada indivíduo e ordenação dos indivíduos por *fitness*.

- **Genético:** é a classe que realiza a evolução da população. Possui os métodos de seleção, recombinação e mutação que agem sobre uma população para a geração de melhores candidatos à solução.

No AG implementado, primeiramente gera-se uma população de n cromossomos aleatórios que representam os circuitos hamiltonianos, ou seja, as possíveis soluções para o problema. A distância total dos primeiros roteiros e seus respectivos *fitness* são calculados logo em seguida. Para este trabalho, a função objetivo adotada corresponde à equação 3.1:

$$fitness_i = d(n) - d(i) + 10 \tag{3.1}$$

em que $d(n)$ corresponde a distância da rota mais longa presente na população, $d(i)$ a distância da rota avaliada. A constante 10 é utilizada para garantir que o processo de seleção ocorra corretamente. Pode ocorrer, em casos extremos, que todos os indivíduos da população tornem-se iguais durante o processo evolutivo. Sendo assim, a constante é utilizada para garantir que todos os indivíduos da população possuam o mínimo de aptidão para serem selecionados.

Também foi discutida a possibilidade de se utilizar como função *fitness* a equação $fitness_i = \frac{1}{d(i)}$. Entretanto, em alguns testes preliminares, a Equação 3.1 apontou uma convergência mais rápida das distâncias dos roteiros e, por isso, foi utilizada neste trabalho.

O cálculo do *fitness* considera a distância da pior rota e esta varia a cada nova população. Assim, o melhor cromossomo de duas populações, mesmo sendo iguais, podem apresentar *fitness* distintos. Esta função de *fitness* converte o problema de minimização, encontrar a menor rota, em um problema de maximização, que é mais intuitivo quando se deseja conferir maior probabilidade de seleção aos indivíduos mais aptos da população.

Além disso, há outro cálculo realizado a cada iteração que corresponde ao cálculo da distância total de cada roteiro, como mencionado anteriormente. A distância entre duas cidades é dada pelo cálculo da distância euclidiana entre dois pontos, conforme a equação 3.2:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{3.2}$$

em que a tupla (x, y) corresponde às coordenadas das cidades.

O primeiro passo antes de iniciar o processo de evolução consiste em preservar o melhor indivíduo da população atual (elitismo). Então, dá-se início ao processo de evolução.

O operador de seleção escolhido para este trabalho foi o método de **seleção por giro de roleta**, descrito na Seção 2.3.3. Outros métodos de seleção não foram utilizados neste trabalho. Os pais são escolhidos aleatoriamente e os cruzamentos ocorrem até que seja gerada uma nova população. O processo de recombinação implementado consiste no **crossover OX**, descrito na Seção 2.3.4. Os pontos de corte do método de recombinação são escolhidos aleatoriamente, em que os pontos selecionados se diferem entre si. Também existe o parâmetro taxa de reprodução que define a probabilidade de ocorrência da recombinação. Este teste ocorre toda vez que os dois pais são selecionados. Após a geração de cada filho ocorre o teste que verifica se ocorrerá **mutação**. Cada filho gerado poderá sofrer mutação, considerando o valor do parâmetro taxa de mutação empregado no algoritmo. Quando ocorrer mutação, duas posições são escolhidas aleatoriamente e invertidas na rota, conforme descrito com mais detalhes na Seção 2.3.5.

O processo é repetido até que se atinja o critério de parada. Quando este critério é atingido, a melhor rota encontrada, seu *fitness* e o número de gerações produzidas são apresentados.

O algoritmo genético clássico foi implementado conforme o Algoritmo 1, apresentado na Seção 2.1.

3.3 Estratégias *delete-cross*

Conforme apresentado anteriormente, o operador *delete-cross* possui como principal característica a remoção de cruzamentos presentes no roteiro do caixeiro viajante, acelerando a convergência do AG. Por se tratar de um operador adicional, ele é inserido no processo de evolução e passa a atuar sobre os cromossomos filhos gerados no processo de reprodução. Sendo assim, para avaliar a eficácia deste método, duas abordagens foram adotadas na implementação dos algoritmos que incluem a heurística.

A primeira abordagem consiste basicamente na implementação do operador descrito no Algoritmo 2 da Seção 2.4. A vantagem deste primeiro método é a rápida convergência quando a população de cromossomos possui um número elevado de cruzamentos em sua rota. Em contrapartida, sua complexidade computacional é alta. O algoritmo do *delete-cross* percorre toda a nova população, cidade por cidade, e verifica se uma aresta formada pelos pontos i e $i + 1$ se cruza com outra aresta formada pelos pontos j e $j + 1$. Isso faz com que a execução de casos de teste com espaço de busca elevado seja demorada, mas com resultados finais satisfatórios.

Neste trabalho, resultados são considerados satisfatórios quando a diferença percentual entre as distâncias obtidas e as mínimas registradas na TSPLIB são menores que 5%, em problemas com até 280 cidades, e 10% para problemas com mais de 280 cidades.

Tendo em vista a complexidade computacional da primeira abordagem, foi implementado um segundo procedimento (denotado por **1-*delete-cross*** neste trabalho) que consiste na substituição do operador de mutação pelo operador *delete-cross* com o objetivo de acelerar a convergência diminuindo o consumo de tempo. Todavia, a função da heurística nesta abordagem é remover apenas um cruzamento dos descendentes de uma geração.

O pseudocódigo que explica o funcionamento da implementação da estratégia *delete-cross* inserido ao AG típico é detalhado no Algoritmo 3.

Algoritmo 3: AG COM OPERADOR *delete-cross*

```
gerar uma população aleatória de cromossomos (candidatos a solução do problema)
calcular o fitness de cada indivíduo na população
repita
    selecionar um par de cromossomos da população atual, através do operador de
      seleção
    através do operador de recombinação, realizar o cruzamento dos pais para gerar
      novos filhos
    realizar a mutação dos novos filhos
    encontrar e eliminar todas as interseções entre arestas
    avaliar o fitness dos novos indivíduos gerados
    substituir os indivíduos considerados inaptos na população
até que o critério de parada seja atendido;
```

O algoritmo do operador *delete-cross* foi inserido após a etapa de mutação, conforme descrito na Seção 2.4. Essa etapa é executada no processo evolutivo de todos os indivíduos da população em todas as evoluções, garantindo que novos cruzamentos gerados através do processo evolutivo sejam removidos imediatamente.

Em relação ao operador 1-*delete-cross*, o Algoritmo 4 explana o funcionamento do mesmo.

Algoritmo 4: AG COM OPERADOR 1-*delete-cross*

gerar uma população aleatória de cromossomos (candidatos a solução do problema)
calcular o *fitness* de cada indivíduo na população
repita
  selecionar um par de cromossomos da população atual, através do operador de seleção
  através do operador de recombinação, realizar o cruzamento dos pais para gerar novos filhos
  eliminar a primeira interseção entre arestas encontrada
  avaliar o *fitness* dos novos indivíduos gerados
  substituir os indivíduos considerados inaptos na população
até *que o critério de parada seja atendido*;

Seu funcionamento é similar ao Algoritmo 3. Sua diferença consiste na remoção da etapa de mutação, além da regra para eliminação das interseções entre arestas. Enquanto no Algoritmo 3 todos os cruzamentos são removidos, no Algoritmo 4 apenas a primeira interseção identificada é removida.

3.3.1 Detecção de Cruzamentos entre Arestas

O algoritmo do *delete-cross* apresentado na Seção 2.4 não possui nenhum método específico para a detecção de arestas cruzadas. Para realizar esta verificação, Cormen *et. al.* [25] apresentam um método com complexidade de tempo $O(1)$ que computa a orientação relativa de um segmento, calculada através do produto vetorial utilizando três pontos. Isso elimina a necessidade de computar o ângulo formado por esses segmentos. A Figura 3.2, adaptada de Cormen *et. al.* [25] mostra como essa descoberta ocorre.

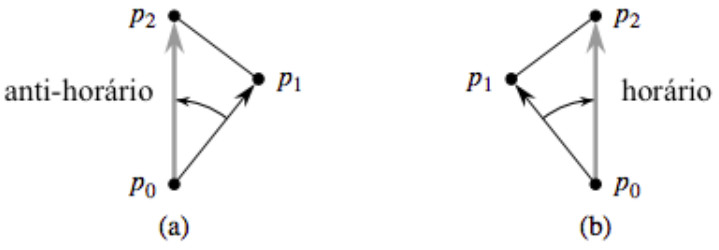

Figura 3.2: Orientações relativas dos segmentos formados por 3 pontos.

Na Figura 3.2 (a), o produto vetorial $(p_1 - p_0 \times p_2 - p_0)$ é negativo. Isso significa que o segmento $\overline{p_1 p_2}$ forma um ângulo em sentido anti-horário de $\overline{p_0 p_2}$. Em (b), o resultado de

produto vetorial é positivo e, consequentemente, o ângulo formado de $\overline{p_1p_2}$ para $\overline{p_0p_2}$ é no sentido horário.

Possuindo essa informação, é possível verificar se ocorre a interseção entre dois segmentos de reta conforme exemplificado na Figura 3.3 [25].

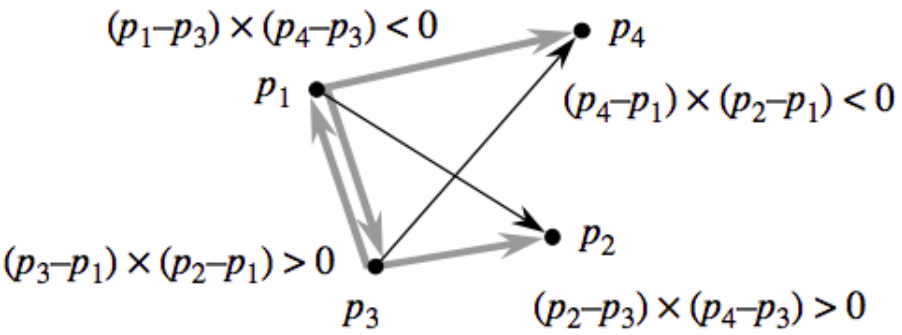

Figura 3.3: Verificação da interseção entre dois segmentos de reta através do produto vetorial

Ao calcular os produtos vetoriais $(p_3 - p_1 \times p_2 - p_1)$ e $(p_4 - p_1 \times p_2 - p_1)$, nota-se que os sinais de ambos diferem entre si. Isso significa que os vetores normais a p_3, $\overline{p_3p_1}$ e $\overline{p_3p_2}$, possuem direções opostas. Similarmente, o sinal do produtos vetoriais $(p_1 - p_3 \times p_4 - p_3)$ e $(p_2 - p_3 \times p_4 - p_3)$ também diferem, significando que os segmentos $\overline{p_1p_3}$ e $\overline{p_1p_4}$ possuem, igualmente, direções opostas em relação ao ponto p_1. Sendo assim, pode-se concluir que os dois segmentos se cruzam.

Os detalhes de funcionamento desta etapa de detecção de cruzamento entre arestas são dados pelo Algoritmo 5.

Algoritmo 5: DETECÇÃO DE CRUZAMENTO ENTRE ARESTAS

Entrada: os quatro pontos a serem analisados, p_1, p_2, p_3, p_4
Saída: $VERDADEIRO$ caso ocorre cruzamento, $FALSO$ caso não ocorra
d_1 = $(p_1 - p_3 \times p_4 - p_3)$
d_2 = $(p_2 - p_3 \times p_4 - p_3)$
d_3 = $(p_3 - p_1 \times p_2 - p_1)$
d_4 = $(p_4 - p_1 \times p_2 - p_1)$
se *(*$(d_1 > 0$ ***e*** $d_2 < 0)$ ***ou*** *(*$d_1 < 0$ ***e*** $d_2 > 0)$*)* ***e*** *(*$(d_3 > 0$ ***e*** $d_4 < 0)$ ***ou*** *(*$d_3 < 0$***e***$d_4 < 0)$*)*
então
  retorne $VERDADEIRO$
fim
senão
  retorne $FALSO$
fim

No algoritmo, as variáveis d_1, d_2, d_3 e d_4 são responsáveis por armazenar os produtos vetoriais referentes aos pontos das duas arestas avaliadas no processo de detecção de cruzamentos. A condicional verifica se os vetores normais aos pontos avaliados possuem direções opostas, apontado o cruzamento entre arestas.

Para cada par de arestas avaliadas o método é executado. Então, quando a interseção é encontrada, o operador a remove. Portanto, a complexidade do método *delete-cross* torna-se $O(n^2)$.

3.4 Critério de Parada

O critério de parada para os três algoritmos testados neste trabalho é definido com base na ideia de que instâncias maiores do PCV necessitarão de mais gerações estáticas para evoluir a solução. Para isso, foi definido uma função de crescimento logarítmico para garantir que grandes casos de teste evoluam satisfatoriamente antes de pararem. O resultado da função é utilizado como o número de gerações consecutivas sem melhora para a parada do algoritmo.

A função é dada por:

$$NGE = (C - n) \cdot e^{\frac{n}{400}} \tag{3.3}$$

em que NGE é igual ao número de gerações estáticas, C é uma constante dada pelo usuário para ajustar o crescimento da função, e é a constante de Euler e n se refere ao número de cidades do roteiro.

3.5 Ambiente Computacional

Como mencionado na Seção 3.2, os Algoritmos Genéticos foram implementados em linguagem Java utilizando o ambiente de desenvolvimento NetBeans IDE 8.1. As características da arquitetura utilizada para implementação e execução das diferentes abordagens são:

- **Processador:** Intel Core i7-3537U, 2.00GHz
- **Memória (RAM):** 6.00 GB
- **Sistema Operacional:** 64 bits, processador base em x64

A parametrização dos algoritmos, bem como as definições relacionadas aos experimentos e os resultados obtidos utilizando os arquivos de teste mencionados na Seção 3.1 serão apresentados no Capítulo 4.

Capítulo 4

Experimentação e Resultados

Os experimentos realizados neste trabalho consistem em coletar, analisar e comparar os resultados produzidos por três estratégias propostas (AG, AG com *delete-cross* e AG com 1-*delete-cross*) no âmbito da qualidade das soluções e do número de gerações reproduzidas para obter os resultados.

Foram realizados dez testes em cada estratégia para cada caso de teste simétrico (sTSP) proveniente da TSPLIB, conforme citado na Seção 3.1. A seguinte seção apresenta os parâmetros que foram utilizados para executar os testes, bem como as definições acerca do critério de parada. A Seção 4.2 apresenta os resultados obtidos com os experimentos realizados.

4.1 Parametrização dos Testes

Para a realização dos testes dos algoritmos implementados, foram definidos os parâmetros mencionados na Seção 3.2. Os parâmetros utilizados para a execução do AG são:

- **Tamanho da população:** igual ao número de cidades do roteiro;
- **Taxa de reprodução:** probabilidade de 90% de ocorrer recombinação;
- **Taxa de mutação:** probabilidade de 1% (com exceção no 1-*delete-cross* onde não ocorre);
- **Número de cromossomos na elite:** apenas o melhor da população atual;

Em relação ao critério de parada definido na Seção 3.4, foi estabelecido que a constante C fosse igual a 1791. Este valor foi definido experimentalmente e visa, tão somente, permitir que

resultados pudessem ser obtidos em tempo hábil para redação deste trabalho. Na formulação original, $NGE = 500 * e^{(n/400)}$, onde n corresponde ao número de cidades da rota.

4.2 Resultados Obtidos

Nesta seção serão apresentadas as médias das soluções dos problemas analisados com as três estratégias, bem como os melhores resultados obtidos e suas diferenças para os valores ótimos da TSPLIB. Além disso, será mostrado um comparativo das estratégias considerando a evolução de apenas uma geração, evidenciando seus efeitos sobre a evolução das soluções e o tempo de execução.

A Tabela 4.1 apresenta as médias dos resultados e do número de gerações para as soluções encontradas pelos algoritmos implementados. As médias dos resultados foram calculadas por:

$$media = \frac{\sum_{i=1}^{t} resultado(i)}{t} \tag{4.1}$$

Resumidamente, o somatório dos resultados obtidos dividido pelo número de testes. As distâncias finais, bem como o número de evoluções executadas para se obtê-las, de todos os testes realizados estão individualmente disponíveis no Apêndice A.

Tabela 4.1: Médias dos resultados e do número de gerações na solução de diferentes PCV

Informações			AG Clássico		1-*delete-cross*		*delete-cross*	
Arquivo	Nº de cidades	Solução ótima	Resultado	Gerações	Resultado	Gerações	Resultado	Gerações
eil51	51	426	449,9	9604,5	439,3	6567,3	430,6	4962,9
kroA100	100	21282	27213,2	27234,6	23934,6	14422,1	21517	8840,3
a280	280	2579	5601,2	132751,9	3833,6	54010,3	2768,8	23664,2
rd400	400	15281	39569	217230,1	23662,2	38836,6	16289,4	66269,5
d657	657	48912	164666,5	258540,4	85307,1	59308,4	53733	103020,3

A Tabela 4.2 apresenta a diferença, em percentual, entre o resultado obtido nos experimentos e o resultado ótimo da TSPLIB nas três abordagens analisadas. Essa diferença é calculada a partir da Equação 4.2.

$$diferenca[\%] = \frac{media - otimo}{otimo} \cdot 100 \tag{4.2}$$

Em que $otimo$ se refere ao valor ótimo registrado na TSPLIB.

Ambas as formulações apresentadas anteriormente são utilizadas para calcular os resultados obtidos exibidos nas Tabelas 4.3 e 4.4.

No que se refere à qualidade das soluções, é possível notar que o AG clássico proporciona bons resultados (diferença menor que 10%) para instâncias pequenas do PCV. Todavia,

Tabela 4.2: Diferença em percentual entre a média dos resultados e o resultado ótimo da TSPLIB

Arquivo	Solução ótima	AG clássico (%)	1-*delete-cross* (%)	*delete-cross* (%)
eil51	426	5,6%	3,1%	1,1%
kroA100	21282	27,8%	12,4%	1,1%
a280	2579	117,1%	48,6%	7,3%
rd400	15281	158,9%	54,8%	6,6%
d657	48912	236,6%	74,4%	9,8%

quando utilizado para instâncias maiores que 100 cidades, seus resultados não foram satisfatórios, apresentando rotas mais longas. Para 100 cidades a rota gerada é 27,8% mais longa que a rota mínima, enquanto na instância com 657 cidades, essa diferença chega a 236,6%. A estratégia 1-*delete-cross* obtém rotas menores que o AG clássico e em um número menor de gerações; entretanto, consideradas insatisfatórias para as instâncias a partir de 100 cidades. Sobre o *delete-cross* é possível notar que as rotas encontradas são satisfatórias para todos os casos de teste.

Quanto ao número de gerações, percebe-se que o *delete-cross* é mais eficaz que as outras estratégias. Por exemplo, para o caso de 657 cidades, empregou pouco menos da metade do número de gerações (165782) quando comparado ao AG clássico (369355).

Também são apresentadas as melhores soluções obtidas, o número de gerações exato para a solução e o erro em relação ao ótimo da TSPLIB nos testes entre as três abordagens analisadas através das Tabelas 4.3 e 4.4.

Tabela 4.3: Melhores resultados obtidos e número de gerações exatas para a solução

Informações			AG Clássico		1-*delete-cross*		*delete-cross*	
Arquivo	Nº de cidades	Solução ótima	Resultado	Gerações	Resultado	Gerações	Resultado	Gerações
eil51	51	426	439,5	11551	430,8	7678	428,8	7770
kroA100	100	21282	25691,9	20149	22309,3	22537	21285,4	8847
a280	280	2579	5009,5	151235	3411,8	86118	2690,3	45759
rd400	400	15281	32301,0	330182	22137,8	69956	15949,3	68609
d657	657	48912	136229,7	369355	80581,6	63578	51899,3	165782

Tabela 4.4: Diferença em percentual entre o melhor resultado obtido e o resultado ótimo da TSPLIB

Arquivo	Solução ótima	AG clássico (%)	1-*delete-cross* (%)	*delete-cross* (%)
eil51	426	3,1%	1,1%	0,6%
kroA100	21282	20,7%	4,8%	0,01%
a280	2579	94,2%	32,2%	4,3%
rd400	15281	111,3%	44,8%	4,3%
d657	48912	178,5%	64,7%	6,1%

As observações válidas para as soluções médias repetem-se para as melhores soluções individuais obtidas. As soluções alcançadas pela estratégia *delete-cross* são melhores que as outras abordagens, inclusive mantendo uma diferença inferior a 10% em relação à solução ótima.

Além disso, para avaliar a eficácia do operador *delete-cross* perante às outras abordagens, foram realizados dez testes que mostram a evolução na primeira geração. A Tabela 4.5 expõe essa informação.

Tabela 4.5: Melhoria média da solução na execução de uma geração

Arquivo	AG Clássico			1-*delete-cross*			*delete-cross*		
	$d_{inicial}$	d_{final}	Melhora	$d_{inicial}$	d_{final}	Melhora	$d_{inicial}$	d_{final}	Melhora
eil51	1395,7	1374,8	1,49%	1454,7	1408,8	3,26%	1394,1	651,1	114,11%
kroA100	14871,3	146317,9	1,61%	151727,6	146711,4	3,42%	147819,7	45271,4	226,52%
a280	31217,7	31035,9	0,58%	31282,6	31108,8	0,56%	31500,7	6071,2	418,85%
rd400	198702,8	197683,9	0,51%	199430,2	196967,9	1,25%	199733,3	36944,6	440,63%
d657	814244,7	809568,7	0,57%	810974,7	806234,5	0,59%	814672,6	120042,8	578,65%

Observa-se que a evolução obtida com o operador *delete-cross* é muito superior às demais estratégias. Ao remover todos os cruzamentos das rotas iniciais o comprimento da menor rota sofre uma redução entre 114,11% a 578,65%, enquanto as outras estratégias variam entre 0,51% e 3,42%.

Utilizando a arquitetura apresentada na Seção 3.5, foram medidos os tempos de execução das 10 primeiras gerações de cada um dos casos de teste, com o objetivo de comparar, entre as três abordagens, o tempo médio de uma evolução da população. Os resultados são exibidos na Tabela 4.6.

Tabela 4.6: Média do tempo (em milissegundos) de uma evolução nas 10 primeiras gerações

Arquivo	AG clássico (ms)	1-*delete-cross* (ms)	*delete-cross* (ms)
eil51	7	15	30
kroA100	14	28	75
a280	52	61	482
rd400	97	112	918
d657	268	278	3248

Ao analisar os tempos de execução dos algoritmos, é notável o quanto o tempo de evolução aumenta ao inserir o operador *delete-cross* no AG. Isso se deve à complexidade do método, explicada na Seção 3.3. Entretanto, não há um aumento considerável no tempo de execução ao comparar o AG clássico com a estratégia 1-*delete-cross*.

É importante ressaltar que o tempo empregado pelo *delete-cross* nas primeiras evoluções é maior, pois é quando o algoritmo realizará o maior número de trocas entre os vértices do roteiro para remover os cruzamentos entre as arestas. A tendência é que, ao reduzir o número de cruzamentos, reduz-se o número de trocas a serem realizadas e consequentemente o tempo de execução do operador.

Para evidenciar como o método de remoção de cruzamentos age sobre os roteiros do caixeiro viajante, as Figuras 4.1 e 4.2 mostram, utilizando a interface visual da aplicação, a evolução da primeira para a décima geração do caso de teste eil51 quando o operador é empregado.

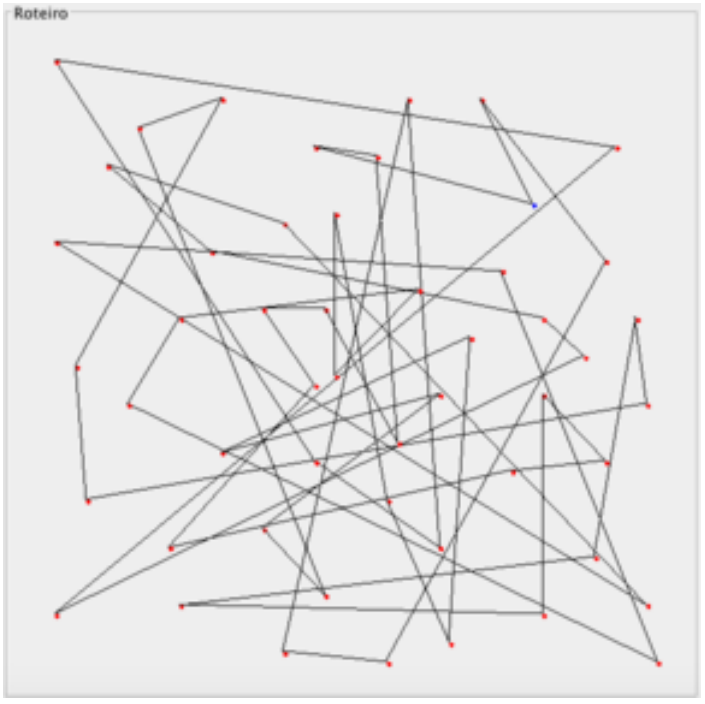

Figura 4.1: Melhor indivíduo da primeira geração no caso de teste eil51

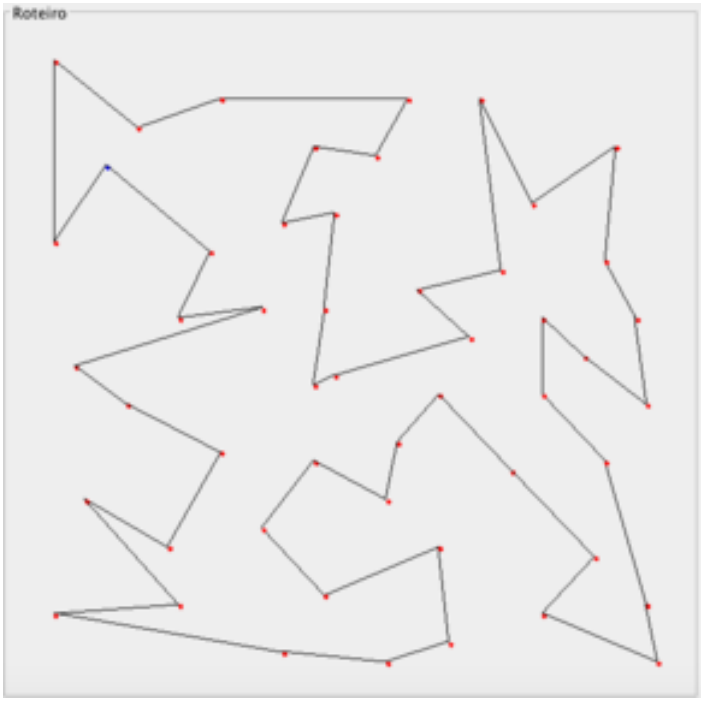

Figura 4.2: Melhor rota da décima geração no caso de teste eil51

Capítulo 5

Considerações Finais e Sugestões para Trabalhos Futuros

O Problema do Caixeiro Viajante é um problema que possui grande complexidade de tempo. Sua resolução é amplamente estudada por diversas áreas da Ciência da Computação e, como mencionado na revisão bibliográfica, existem diversos métodos que possuem o objetivo de solucioná-lo. O Algoritmo Genético é uma meta-heurística que pode ser utilizada para esse fim e métodos específicos podem ser incorporados a ele para a obtenção de melhores resultados, como é o caso do operador *delete-cross*.

Os experimentos realizados neste trabalho mostram que o emprego do operador *delete-cross* trouxe resultados satisfatórios para todas as instâncias testadas do PCV, produzindo inclusive menos gerações que as outras estratégias abordadas. Entretanto, foi visto que seu custo computacional é alto e que isso impacta diretamente no tempo de execução do AG. Ou seja, um Algoritmo Genético clássico pode produzir mais gerações em menos tempo do que quando empregado o operador *delete-cross*. Então, na tentativa de amenizar o impacto no tempo de execução, foi proposto e testado o operador 1-*delete-cross*, que obteve resultados até 4 vezes melhores que o AG clássico, porém, insatisfatórios para grandes instâncias do problema.

5.1 Trabalhos Futuros

Tendo em vista a complexidade de tempo quando o operador *delete-cross* é incorporado ao AG, aconselha-se medir o tempo de execução entre as três abordagens. Outra abordagem que também pode ser utilizada para diminuir o impacto do operador no processo evolutivo em relação ao tempo de convergência é aplicar a remoção dos cruzamentos a cada n gerações.

Outra sugestão seria avaliar se há influência no tempo de convergência e na qualidade das soluções ao utilizar métodos distintos de seleção, como a seleção por torneio. É evidente que o método de seleção por giro de roleta faz com que o indivíduo elitista tenha grande influência sobre a variabilidade genética da população, além de acelerar a convergência para um mínimo local, visto que o mesmo possui maior probabilidade de se reproduzir.

Além do operador de seleção, aconselha-se avaliar outros métodos de recombinação quando aplicado o operador *delete-cross* em um AG, além do operador OX usado neste trabalho.

Outro ponto a analisar refere-se ao impacto da distância entre os pontos de corte aleatorizados pelo operador OX, pois quanto maior a distância entre os pontos, maior a semelhança entre os pais e os filhos gerados, reduzindo a variabilidade genética da população. Espera-se que a redução da distância entre os pontos de corte, aumente a diversidade genética da população.

Neste trabalho, o operador 1-*delete-cross* elimina sempre o primeiro cruzamento encontrado em uma rota. Outra sugestão para um trabalho futuro seria alterá-lo para que seja sorteado uma cidade do roteiro e, a partir dessa cidade, remover o primeiro cruzamento encontrado, trazendo aleatoriedade para o método.

Por fim, recomenda-se realizar uma análise estatística não paramétrica para testar a igualdade de médias entre as soluções e o número de gerações das diferentes abordagens implementadas, nos diversos casos de testes empregados.

Apêndice A

Resultados Detalhados dos Testes

A.1 Testes Individuais das Estratégias Analisadas

Tabela A.1: Testes executados com a estratégia *delete-cross*

eil51		kroA100		a280		rd400		d657	
Distância	Gerações	Distância	Gerações	Distância	Gerações	Distância	Gerações	Distância	Gerações
430,24	5608	21358,77	8593	2690,32	45759	16230,12	67671	53812,59	102526
433,97	4290	21577,08	11311	2902,20	12570	16711,55	51551	55135,8	61835
430,24	4667	21544,24	8973	2763,45	27181	16232,66	64129	53424,1	135327
430,74	6867	21285,44	8847	2775,07	22462	16328,01	64610	52853,93	115620
431,17	3272	22050,75	7679	2726,64	16792	16063,44	72124	53981,34	98546
430,24	4115	21381,83	12869	2693,07	25270	16550,55	51923	51899,30	165782
428,87	7770	21285,44	9891	2819,21	20533	16300,06	63987	54110,44	76501
433,21	5075	21518,95	6174	2769,93	18503	16207,14	87887	53801,98	87397
428,98	2853	21599,06	8262	2773,54	16512	16321,14	70204	55749,50	62091
428,98	5112	21568,92	5804	2775,19868	31060	15949,37	68609	52561,11	124578
430,66	4962,9	21517,05	8840,3	2768,86	23664,2	16289,41	66269,5	53806,63	103827

Tabela A.2: Testes executados com a estratégia 1-*delete-cross*

eil51		kroA100		a280		rd400		d657	
Distância	Gerações	Distância	Gerações	Distância	Gerações	Distância	Gerações	Distância	Gerações
437,90	5493	22753,56	12573	3945,11	50107	24321,86	38700	83972,27	92808
445,20	8762	24546,05	25164	3921,21	42781	25644,84	19410	82517,31	102096
442,12	5047	25039,53	7590	3958,57	48349	24287,11	33484	80581,59	63578
447,09	3977	23807,68	11269	3690,31	82659	22722,53	27669	83394,41	81109
430,89	7678	23043,78	20154	3739,82	51051	22137,82	69956	84936,56	39270
440,49	4313	23696,89	9248	3411,84	86118	22860,15	27236	87482,48	50644
437,39	9645	26589,46	7796	3936,20	45755	23913,06	44740	89779,63	34450
440,58	7655	23568,59	13103	3964,07	37219	23474,25	38286	86199,42	44439
435,82	6914	23992,08	14787	3913,71	57493	23528,42	47661	89225,61	42319
435,64	6189	22309,32	22537	3856,02	38571	23732,43	41224	84981,41	42371

Tabela A.3: Testes executados com o AG clássico

eil51		kroA100		a280		rd400		d657	
Distância	Gerações	Distância	Gerações	Distância	Gerações	Distância	Gerações	Distância	Gerações
445,74	7256	28163,68	30268	6321,97	84587	41437,72	200452	136229,76	369355
445,23	11454	25735,76	31370	5519,52	136619	41098,93	197672	166632,81	266498
443,04	12886	25795,71	24178	5677,64	125110	36107,11	287556	190332,47	202032
457,28	12772	27430,22	32036	5892,56	118547	38506,96	218116	175471,26	230295
463,05	6270	26578,79	26761	5622,96	158343	32301,02	330182	168213,89	282695
447,99	10058	29861,77	25913	5471,17	143199	35525,71	276668	171107,85	197782
439,57	11551	27781,24	27920	5546,31	136090	44530,81	155024	161856,41	266457
455,29	8103	25691,94	20149	5775,69	116027	39083,23	210980	152369,75	271187
446,37	5416	27876,35	21166	5175,20	157762	40854,74	181674	176467,52	197147
455,42	10279	27217,44	32585	5009,59	151235	46244,55	113977	147983,62	301956

Referências Bibliográficas

[1] SOUZA, G. R. de. *Uma Abordagem por Nuvem de Partículas para Problemas de Otimização Combinatória*. Dissertação (Mestrado) — UFRN – Universidade Federal do Rio Grande do Norte, Natal - RN, Fevereiro 2006.

[2] ZAPELINI, C. Z. *Um Estudo Abrangente sobre Metaheurística, incluindo um Histórico*. Dissertação (Trabalho de Conclusão de Curso) — USP – Universidade de São Paulo, São Paulo - SP, Dezembro 2009. Monografia.

[3] MATAI, R.; SINGH, S.; MITTAL, M. L. *Traveling Salesman Problem: An Overview of Applications, Formulations, and Solution Approaches*. Consultado na Internet: https://www.intechopen.com/books/traveling-salesman-problem-theory-and-applications/traveling-salesman-problem-an-overview-of-applications-formulations-and-solution-approaches. Acessado em: 14/11/2017.

[4] KATAYAMA, K.; SAKAMOTO, H.; NARIHISA, H. The efficiency of hybrid mutation genetic algorithm for the travelling salesman problem. *Mathematical and Computer Modelling*, Elsevier, Ltd, v. 31, n. 10, p. 197–203, 2000.

[5] THAMILSELVAN, R.; BALASUBRAMANIE, D. P. A genetic algorithm with a tabu search (gta) for traveling salesman problem. *International Journal of Recent Trends in Engineering and Technology*, v. 1, n. 1, p. 605–608, Novembro 2009.

[6] LAARHOVEN, P. V.; AARTS, E. H. L. *Simulated Annealing: Theory and Applications*. 3. ed. [S.l.]: Kluwer Academic Publishers, 1987.

[7] KENNEDY, J.; EBERHART, R. Particle swarm optimization. *IEEE International Conference on*, v. 4, p. 1942–1948, Agosto 1995.

[8] DORIGO, M.; GAMBARDELLA, L. M. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, v. 1, n. 1, p. 53–66, Abril 1997.

[9] RANI, K.; KUMAR, V. Solving travelling salesman problem using genetic algorithm based on heuristic crossover and mutation operator. *IMPACT: International Journal of Research in Engineering and Technology*, v. 2, n. 2, p. 27–34, Fevereiro 2014.

[10] SHI, X. H. et al. Particle swarm optimization-based algorithms for tsp and generalized tsp. *Information processing letters*, v. 103, n. 5, p. 169–175, Março 2007.

[11] HOLLAND, J. H. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. Cambridge, MA: MIT Press, 1975.

[12] LUCAS, D. C.; ÁLVARES, L. O. *Algoritmos Genéticos: uma Introdução*. Porto Alegre - RS, Março 2002.

[13] MORAIS, J. L. M. *Problema do Caixeiro Viajante Aplicado ao Roteamento de Veículos numa Malha Viária*. Dissertação (Trabalho de Conclusão de Curso) — UNIFESP - Universidade Federal de São Paulo, São José dos Campos - SP, Dezembro 2010. Monografia.

[14] GANHOTO, M. A. *Abordagens para problemas de roteamento*. Tese (Doutorado) — UNICAMP - Universidade Estadual de Campinas, Campinas - SP, 2004.

[15] MITCHELL, M. *An Introduction to Genetic Algorithms*. Cambridge, MA: MIT Press, 1996.

[16] LACERDA, E. G. M.; CARVALHO, A. C. P. L. F. Sistemas inteligentes: aplicações a recursos hídricos e ciências ambientais. In: ____. Porto Alegre - RS: UFRGS: ABRH, 1999. cap. : Introdução aos Algoritmos Genéticos, p. 99–150.

[17] BENEVIDES, P. F. *Aplicação de heurísticas e metaheurísticas para o problema do caixeiro viajante em um problema real de roterização de veículos*. Dissertação (Mestrado) — UFPR - Programa de Pós-Graduação em Métodos Numéricos em Engenharia, Curitiba - PR, 2012.

[18] GOLDBERG, D. E.; LINGLE, J. R. Alleles, loci and the tsp. In: GREFENSTETTE, J. J. (Ed.). *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*. Hillsdale, New Jersey: [s.n.], 1985. p. 154–159.

[19] OLIVER, I. M.; SMITH, D.; HOLLAND, J. R. C. A study of permutation crossover operatorson the tsp. In: GREFENSTETTE, J. J. (Ed.). *Proceedings of the Second International Conference on Genetic Algorithms and Their Application*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1987. p. 224–230.

[20] DAVIS, L. Applying adaptive algorithms to epistatic domains. In: *Proceeding of the International Joint Conference on Artificial Intelligence*. Los Angeles, USA: IEEE Computer Society Press, 1985. p. 162–164.

[21] CROES, G. A. A method for solving traveling-salesman problems. Operations Research, Maryland, USA, p. 791–812, 1958.

[22] PAN, G. et al. Hybrid immune algorithm based on greedy algorithm and delete-cross operator for solving tsp. In: *Soft Computing*. [S.l.]: Springer-Verlag Berlin Heidelberg, 2016. v. 20, n. 2, p. 555–566.

[23] WU, J.; OUYANG, A. A hybrid algorithm of aco and delete-cross method for tsp. In: *2012 International Conference on Industrial Control and Electronics Engineering*. Kansas City: IEEE, 2012. p. 1694–1696.

[24] REINELT, G. *TSPLIB*. 1995. Consultado na Internet: http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/. Acessado em 14/11/2017.

[25] CORMEN, T. H. et al. *Introduction to Algorithms*. 3. ed. Cambridge, MA: MIT Press, 1990.