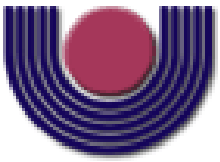

UNIOESTE – Universidade Estadual do Oeste do Paraná

CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
Colegiado de Ciência da Computação

Curso de Bacharelado em Ciência da Computação

Explorando os operadores *crossover* OX e 1-*delete-cross* em Algoritmo Genético aplicado ao Problema do Caixeiro Viajante

Paulo Henrique Tenório

CASCAVEL
2018

PAULO HENRIQUE TENÓRIO

EXPLORANDO OS OPERADORES *CROSSOVER* OX E 1-*DELETE-CROSS* EM ALGORITMO GENÉTICO APLICADO DO PROBLEMA DO CAIXEIRO VIAJANTE

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
2018

PAULO HENRIQUE TENÓRIO

EXPLORANDO OS OPERADORES *CROSSOVER* OX E 1-*DELETE-CROSS* EM ALGORITMO GENÉTICO APLICADO DO PROBLEMA DO CAIXEIRO VIAJANTE

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, 21 de novembro de 2018

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

AG	Algoritmo Genético
CX	*Cyclic Crossover*
OX	*Ordered Crossover*
PCV	Problema do Caixeiro Viajante
PMX	*Partially Mapped Crossover*

Sumário

Resumo

O problema do caixeiro viajante (PCV) consiste em, dado um grupo de cidades, descobrir a menor rota na qual se percorrem todas as cidades uma única vez, retornando à cidade inicial. Embora tenha diversas aplicações práticas, não existe um algoritmo determinístico eficiente para resolver esse problema. Por isso, muitas vezes são usadas técnicas heurísticas para se obter soluções aproximadas em tempo razoável. O Algoritmo Genético (AG) é uma técnica heurística que usa operações baseadas no conceito de seleção natural e, muitas vezes, é combinada com outras heurísticas. Cechinato (2017) elaborou um algoritmo combinando um AG e uma heurística de remoção de cruzamentos, chamada *delete-cross*, que apresentou melhores resultados quando comparados aos resultados fornecidos por um AG clássico. Neste trabalho analisaram-se possíveis melhorias no algoritmo proposto pelo autor. Os resultados de diversos experimentos com o operador de *crossover* OX mostraram que a diversidade populacional, medida pelo coeficiente de variação do comprimento das rotas, não é reduzida durante o processo evolutivo. O operador *delete-cross*, modificado para remover um cruzamento aleatório da rota e não mais o primeiro deles, forneceu resultados entre 2,4% e 18,8% melhores, dependo da instância de PCV resolvido. Ainda em relação ao operador OX, primeiramente avaliou-se o impacto em se gerar intervalos de corte fixos de diversos tamanhos. O corte de tamanho 10% foi o único que se mostrou promissor, porém testes posteriores mostraram que essa solução é inferior se comparado a uma solução com intervalos de corte com tamanhos aleatórios. Entretanto, observou-se que ao usar intervalos de corte menores a convergência do AG é acelerada no início da resolução do problema. Por último, otimizou-se o algoritmo de seleção de intervalos tratando a rota como uma lista circular, permitindo intervalos de corte nas porções extremas da rota, o que melhorou em mais de 15% os resultados para os PCV maiores.

Palavras-chave: aleatorização, diversidade populacional, intervalo de corte, representação cíclica.

Capítulo 1
Introdução

A humanidade ao longo da sua história criou diversas técnicas e tecnologias, e muitas delas foram inspiradas com base no que é observado na natureza. Tanto que, nos anos 50, foram propostos termos para definir tais *designs* baseados na natureza como *Biomimetics* e *Bionics* (VINCENT *et al.*, 2006).

Na área da Computação não é diferente. Diversos algoritmos heurísticos populares foram inspirados em fenômenos naturais, citando alguns exemplos: Otimização por Colônia de Formiga, inspirado no movimento de formigas (YANG *et al.*, 2008); Sistema Imune Artificial, baseado na resposta do sistema imune de vertebrados (GONG, JIAO e ZHANG, 2006); Otimização por Enxame de Partículas, baseado em movimento de grupos de animais, como em cardume de peixes ou bando de pássaros (YANG *et al.*, 2012); etc.

Nos anos 50 e 60 vários cientistas da computação propuseram métodos de otimização baseados no mecanismo de seleção natural proposto por Charles Darwin. Entre as principais técnicas desenvolvidas estão os Algoritmos Genéticos (AG), Estratégias Evolutivas e Programação Evolutiva (MITCHELL, 1998).

Os AGs foram propostos e desenvolvidos por Holland e seus alunos nos anos 60 e 70. Seu princípio elementar é a resolução de problemas por meio da evolução de uma população de indivíduos com base em operadores inspirados na seleção natural como a seleção, o *crossover* e a mutação (WHITLEY, 1994).

No processo de evolução de um AG gera-se uma nova população composta por novos indivíduos, os quais são gerados pela combinação de características dos indivíduos da população anterior. Devido a isso, existe um conceito associado a uma população de um AG chamado variabilidade populacional (MITCHELL, 1998).

Variabilidade populacional é uma medida que indica quão distintos são os indivíduos de uma população. Como novos indivíduos são gerados com base em antigos, é interessante que os membros de uma população sejam distintos entre si, caso contrário um AG dificilmente evoluirá (BANZHAF *et al.*,1998).

O problema do caixeiro viajante (PCV) é um desafio antigo que consiste em encontrar uma rota para visitar todas as *n* cidades de um conjunto, passando por elas uma única vez e no menor percurso possível. O PCV tem diversas aplicações como em perfuração de placas de circuitos integrados, cabeamento de computadores e roteamento de veículos (DAVENDRA, 1996).

O PCV é um problema de natureza combinatorial, por isso a busca exaustiva não é viável para problemas com um número significativo de cidades (n > 30). Por isso o problema mostra-se um terreno fértil para testar soluções heurísticas, ou seja, encontrar boas soluções examinando um número limitado de caminhos. Uma técnica heurística muito usada para resolver esses problemas são os AG (POTVIN, 1996).

Cechinato (2017) testou duas implementações de AG combinados com uma heurística de remoção de cruzamentos, chamada *delete-cross*. A primeira implementação faz uso da heurística para remover todos os cruzamentos encontrados entre arestas de uma rota e obteve bons resultados. Porém, o tempo de processamento na resolução de grandes instâncias do PCV é inviável. A segunda usa a heurística para remover apenas o primeiro cruzamento detectado entre arestas da rota e é rápida, porém, obteve resultados ruins para instâncias do PCV com mais de 100 cidades.

O objetivo deste trabalho foi realizar modificações nos algoritmos propostos por Cechinato (2017), visando melhorar os resultados encontrados pelo autor.

Usando o coeficiente de variação do comprimento das rotas, avaliou-se se os operadores empregados no AG de Cechinato diminuíam a diversidade populacional ao longo do tempo, restringindo a evolução do AG.

Cechinato usou um operador genético de *crossover* especial chamado *Order Crossover* (OX), explicado na seção 2.4. Este operador gera um intervalo de corte entre dois pontos aleatórios do cromossomo. Neste trabalho analisou-se se diferentes tamanhos de corte afetam a qualidade da solução dos PCV. Além disso, foram realizadas melhorias nas implementações de Cechinato. Na primeira delas modificou-se o operador 1-*delete-cross* para remover um cruzamento aleatório entre arestas de uma rota (seção 3.3); na segunda o operador OX foi modificado para considerar a natureza cíclica das rotas manipuladas pelo AG (seção 3.4).

Este trabalho está organizado da seguinte maneira. O capítulo 2 mostra a fundamentação teórica para este trabalho; nela são apresentados os conceitos básicos de computação evolutiva e dos AG, diversidade populacional, PCV e os algoritmos criados por Cechinato

(2017). No capítulo 3 é apresentado o método empregado na análise da diversidade populacional durante a evolução do AG, bem como as melhorias nele realizadas. No capítulo 4 são apresentados os resultados obtidos. Finalmente, no capítulo 5, são apresentadas as conclusões obtidas e as sugestões para trabalhos futuros.

Capítulo 2
Fundamentação Teórica

2.1 Computação Evolutiva e Algoritmos Genéticos

Nos anos 50 e 60 vários cientistas da computação, de modo independente, estudaram os sistemas evolutivos com a ideia de que a evolução poderia ser usada como ferramenta de otimização para problemas da engenharia. A ideia geral desses sistemas era evoluir uma população de soluções candidatas, usando operadores inspirados na seleção natural e variação genética. Essas atividades computacionais, motivadas pela biologia, se dividiram ao longo do tempo e, nos anos 1980, dividiram-se em 3 grandes áreas: redes neurais, aprendizado de máquina e computação evolutiva. Uma das principais áreas da computação evolutiva são os AG criados por Holland nos anos 60 e 70 (MITCHELL, 1998).

Um Algoritmo Genético base possui uma população de soluções candidatas representadas em cromossomos, uma função *fitness* e operadores genéticos como seleção, *crossover* (cruzamento) e mutação. O cromossomo é um valor codificado que representa uma possível solução para o problema em questão. A função *fitness* define o grau de aptidão de cada cromossomo. O operador de seleção é responsável por escolher os cromossomos que poderão se reproduzir, dando prioridade àqueles mais aptos. O operador de *crossover* é responsável por gerar novos indivíduos pela combinação das características herdadas dos cromossomos pais selecionados. Por último, o operador de mutação realiza perturbações aleatórias nos cromossomos, simulando alterações eventuais que acontecem nos cromossomos durante o processo evolutivo (CARR, 2014).

A figura 2.1 ilustra os principais processos que acontecem em um AG. O primeiro processo consiste em se criar uma população aleatória e, na sequência, usando a função de *fitness*, calcular a aptidão de todos os indivíduos da população gerada. Depois se inicia um processo cíclico, constituído pelas operações seleção, *crossover*, mutação e cálculo da aptidão até que o critério de parada seja alcançado. Por exemplo, depois de 100 gerações, ou seja, 100 iterações do processo cíclico.

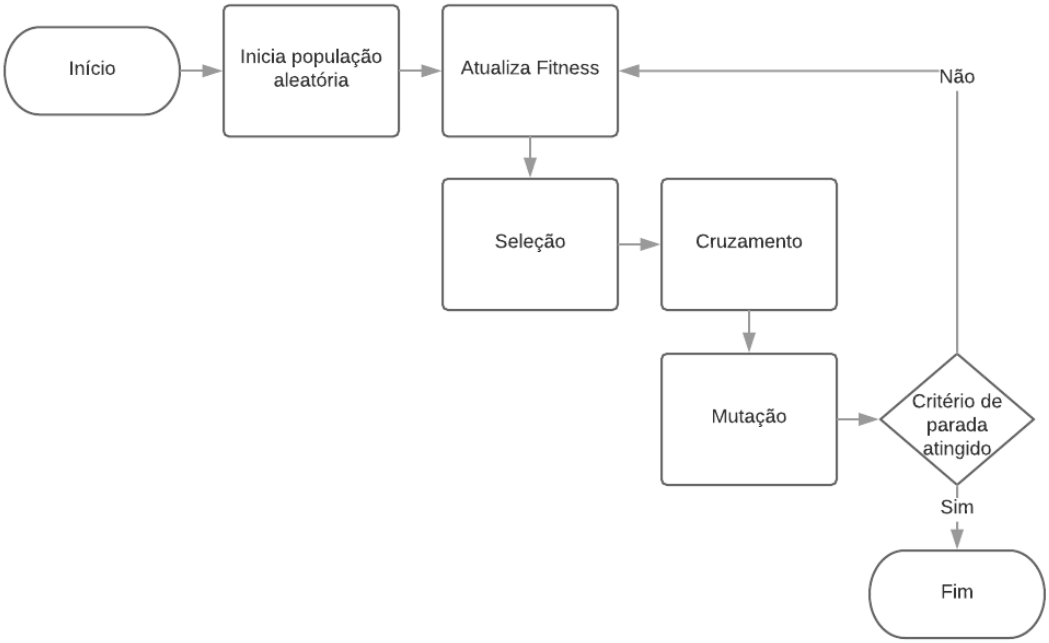

Figura 2.1 – Fluxograma de um AG Clássico

Não há garantia que os filhos gerados nos processos seleção-*crossover*-mutação tenham aptidão maior que seus pais; este processo pode, inclusive, descartar as melhores soluções encontradas em gerações anteriores. Por isso uma adaptação, chamada elitismo, foi desenvolvida por De Jong (1975). Ela tem como finalidade manter os N melhores indivíduos da população corrente copiando-os para a nova população, garantindo que as boas soluções não sejam perdidas (BALUJA e CARUANA, 1995).

Holland também criou o teorema dos esquemas (*Schema theorem*), onde ele tenta explicar o porquê dos algoritmos genéticos funcionarem. Holland argumenta que bons esquemas (sequências de genes 0 e 1) tendem a ser mantidas e compartilhadas entre os indivíduos ao decorrer da evolução da população (MITCHELL, 1998). Ou seja, o algoritmo genético tende a explorar espaços que parecem promissores para a solução (BANZHAF *et al.*, 1998).

Para que um algoritmo genético faça pleno uso de seus operadores, em particular o operador de *crossover*, é interessante que os indivíduos que compõem a população sejam diferentes. Na operação de *crossover*, um cromossomo filho é gerado pela combinação de partes dos cromossomos pais; caso os cromossomos pais sejam semelhantes, o *crossover* resultará um cromossomo filho com poucas mudanças em relação aos progenitores. Assim, este operador não ajudaria na convergência do problema. Populações com pouca diversidade tendem a convergir para um mínimo local (HIEN, *et al*., 2018).

2.2 Diversidade Populacional

Na natureza a diversidade é um fator crucial para a evolução. Isso também é verdade quando se deseja evoluir uma população usando-se algoritmos evolutivos ou algoritmos genéticos, pois a diversidade representa diferenças em comportamento e estrutura dos indivíduos da população. A diversidade populacional é considerada um elemento essencial no estudo dos AG (BURKE, GUSTAFSON e KENDALL, 2004).

A análise da diversidade é útil, pois indica se uma população está evoluindo ou alcançou equilíbrio. Caso se atinja o equilíbrio pode-se parar o processo evolutivo ou então introduzir novos indivíduos, na tentativa de otimizar ainda mais a solução (BANZHAF *et al.*,1998). Os mesmos autores dividem as medidas de diversidade em dois grupos: genotípicas e fenotípicas. As genotípicas medem a diversidade analisando a estrutura do gene. Já as fenotípicas medem a diversidade analisando diferenças de características que se espera estarem relacionadas à diversidade.

Em problemas de otimização de números reais, em geral, prefere-se a análise fenotípica. Quando se trata de análise genotípica todos os bits são tratados igualmente. Porém, a variação de um bit pode significar uma mudança grande em um valor fenotípico. Por exemplo, dois genes 1001 e 0001 têm diferença em apenas um alelo; entretanto, se esses genes codificam um valor inteiro não sinalizado, essa diferença seria de 8 unidades (MORRISON e DE JONG, 2001).

Para quantificar essa diversidade de população existem alguns métodos, que operam sobre a diversidade genotípica ou fenotípica.

2.2.1 Métodos para Avaliação de Diversidade Populacional

A distância de Hamming é uma métrica usada como medida de variabilidade genotípica, representada pelo número de *bits* diferentes de duas *strings*, conforme ilustrado na Figura 2.2.

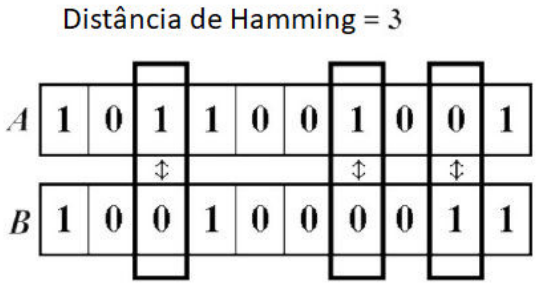

Figura 2.2 – Exemplo de distância de Hamming
Fonte: Adaptado de Ishengoma (2014).

A Figura 2.2 exemplifica o cálculo da distância de Hamming para duas *strings* binárias: A e B. Com esse valor pode se calcular em uma média, ou a maior distância (CORRIVEAU *et al.*, 2012). Neste caso a distância é de 3 unidades, pois existem 3 posições em que as strings diferem. Essa comparação tem que ser realizada entre todos os indivíduos na razão igual a C(n, 2), ou seja, como uma combinação de n indivíduos em grupos de 2, configurando complexidade fatorial (MORRISON e DE JONG, 2001).

Uma segunda métrica empregada na avaliação da diversidade populacional é a entropia. O termo vem da termodinâmica e foi usado primeiro por Clausius em 1865 para interpretar a irreversibilidade em algumas transformações. No século 20 também foi usada como medida de informação. Nesse contexto, a entropia representa a quantidade de desordem em uma população (HIEN *et al.*, 2018) e pode ser estimada pela equação 2.1.

$$E(P) = -\sum_{i}^{k} p_k^{\log(p_k)} \tag{2.1}$$

A população P é particionada em grupos de acordo com o *fitness* e *pk* é a proporção da população que está em determinado grupo. *E*(*P*) é o valor da entropia para a população *P*.

Por exemplo, em uma sala de aula com 25 meninos e 15 meninas, queremos calcular a entropia dessa população (40 estudantes) com base no sexo (masculino ou feminino). A entropia do sistema é o resultado da soma da participação dos dois grupos, chamados aqui de EntropiaMeninos e EntropiaMeninas.

$$Entropia(P) = -(EntropiaMeninos + EntropiaMeninas) \tag{2.2}$$

A contribuição dos meninos e das meninas é calculada por:

$$EntropiaMeninos = 25/40^{\log(25/40)} = 1,1 \tag{2.3}$$

A contribuição das meninas é calculada a seguir:

$$EntropiaMeninas = 15/40^{\log(15/40)} = 1,52 \tag{2.4}$$

Agora calcular a entropia da população somando as contribuições para entropia dos dois grupos.

$$Entropia(P) = -(1,1 + 1,52) = -2,62 \tag{2.5}$$

A entropia mínima é encontrada quando todos os indivíduos pertencem a um mesmo grupo. Digamos que uma sala de aula tivesse apenas meninos. A entropia da população seria resultado da soma de apenas de um grupo, conforme mostrado na equação 2.6.

$$Entropia(P) = -\left(1^{(\log 1)}\right) = -1^0 = -1 \tag{2.6}$$

Assim a entropia mínima de qualquer sistema é -1. Outros trabalhos usam ferramentas estatísticas para avaliar a diversidade populacional com base no fenótipo do indivíduo, em particular quando a dimensão do problema é única. Rosca (1997) faz testes usando a variância do *fitness* para medir a diversidade e Zhu (2003) usa quatro diferentes métricas para analisar a diversidade, sendo uma delas o desvio padrão do *fitness* da população.

2.3 O Problema do Caixeiro Viajante

A Complexidade Computacional é um dos ramos de teoria computacional e da matemática que se preocupa em classificar problemas de acordo com sua dificuldade. Os problemas combinatórios podem ser classificados na classe P ou NP. Problemas da classe P são mais simples, pois para cada problema P existe um algoritmo determinístico de tempo polinomial para resolvê-lo. A classe NP contém os problemas da classe P e outros problemas mais complexos, onde não se conhece uma solução por algoritmo determinístico de tempo polinomial, e não se sabe se tal algoritmo existe. (BOVET e CRESCENZI, 2006).

O problema do caixeiro viajante (PCV) é um desses problemas considerados mais difíceis, pois não se conhece um algoritmo em tempo polinomial para resolvê-lo. Considerado um dos problemas mais estudados na computação matemática (DAVENDRA, 2010).

Nesse problema um caixeiro viajante deve visitar um número de cidades, começar e terminar na mesma cidade percorrendo a menor distância possível. Com o aumento do número de cidades, a determinação da rota ótima (um ciclo hamiltoniano com menor tamanho possível) torna-se extremamente complexa (DAVENDRA, 2010).

A Figura 2.3 exibe um exemplo de PCV resolvido. À esquerda da figura estão representadas as cidades (pontos) que precisam ser visitadas. À direita está representada a rota que possibilita visitar todas as cidades, ou seja, conecta todos os pontos, percorrendo a menor distância possível.

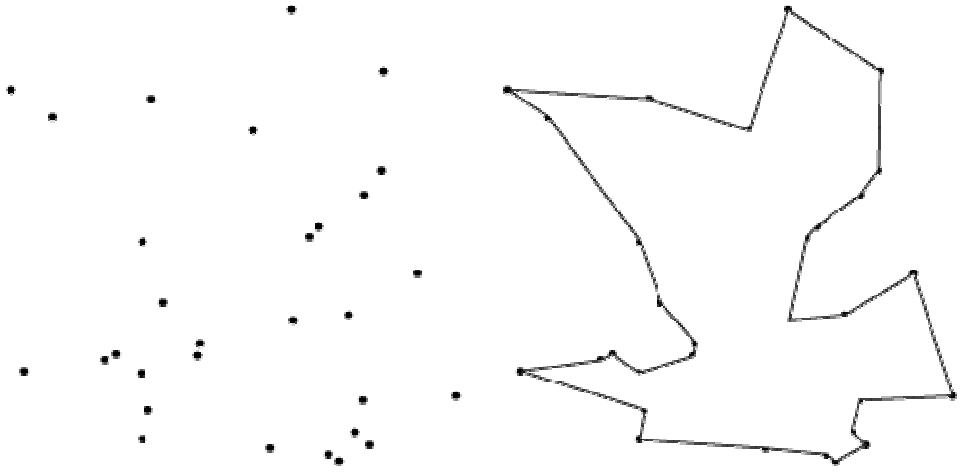

Figura 2.3 – Exemplo de solução de um PCV

Uma das abordagens conhecidas para se resolver o PCV são os algoritmos da computação evolutiva. Alguns exemplos são: Sistema Imune Artificial (GONG, JIAO e ZHANG, 2006), Algoritmos Genéticos (POTVIN, 1996), Colônia de Formigas (YANG *et al.*, 2008), Enxame de Partículas (YANG, 2012), *Self Organizing Migrating Algorithm* (DOKANIA, BAGGA e SHARMA, 2017). Além disso, algoritmos baseados em formulações matemáticas como Busca Tabu (BASU, 2012), Evolução Diferencial (MI *et al.*, 2010) e Scatter Search (ABDULELAH *et al.*, 2017) também demonstraram bons resultados (DAVENDRA, 2010).

Problemas do caixeiro viajante são estudados há muito tempo. Por exemplo Croes, em 1958, propôs uma heurística para acelerar a resolução de problemas PCV chamada de 2-opt. Essa heurística é discutida em mais detalhes seções posteriores. No trabalho de Croes (1958) foi analisado o tempo levado para resolver manualmente alguns problemas do PCV, com poucas cidades, usando ou não a heurística. O problema com maior número de cidades, 42 cidades, levou 70 horas para ser resolvido (CROES, 1958).

Hoje temos um poder computacional muito diferente do que existia para Croes em 1958, então, alguns problemas PCV podem ser resolvidos por testes exaustivos. Porém, existem instâncias do PCV com milhares de cidades e, mesmo hoje, não é viável resolver tais problemas de maneira exaustiva. Entretanto, para esses casos onde não é viável encontrar a distância ótima, o ótimo pode ser aproximado usando o limite inferior de Held-Karp.

2.3.1 Limite Inferior de Held-Karp

O PCV é um problema de otimização combinatória que, para grandes instâncias, não se consegue encontrar a solução ótima em tempo viável. Por isso, muitas vezes, os resultados

obtidos são comparados com o limite inferior de Held-Karp (Held e Karp, 1970; Held e Karp, 1971).

Este limite consiste em uma aproximação do PCV, resolvido por programação linear usando o algoritmo simplex. Resolver esta aproximação não é uma tarefa tão trivial, pois o número de restrições cresce exponencialmente com o número de cidades (ARTS e LENSTRA, 2003).

O limite inferior de Held-Karp não pode ser pior que 66% do ótimo. Porém, em geral, aproxima-se muito mais. Estima-se que em muitos casos aproxima-se de 99.9% ou mais da solução ótima (ARTS e LENSTRA, 2003).

2.4 Adaptações do PCV para Algoritmos Genéticos

Devido a sua formulação simples, o PCV sempre foi um solo fértil para novas ideias. Por isso, é natural que já tenham sido usados algoritmos genéticos para solucionar o PCV. Porém, a abordagem original desenvolvida por Holland não foi projetada para resolver este tipo de problema de otimização combinatória. Naquela época, os domínios abrangidos eram o aprendizado de tarefas e otimização de funções. Por isso devem ser feitas modificações no algoritmo de Holland para adaptá-lo ao PCV. O trabalho de Potvin (1996) apresenta as diversas mudanças propostas para adaptar o PCV para os algoritmos genéticos. As próximas seções apresentam um resumo das modificações básicas.

2.4.1 Codificação dos Cromossomos

Em vez de se representar o cromossomo por uma cadeia binária de zeros e uns, o cromossomo é representado por uma string com a ordem da visita das cidades, ou seja, pela própria rota associada a um indivíduo da população. Por exemplo, em um PCV com 3 cidades, um cromossomo pode ter as possíveis representações cromossômicas 012, 021, 120, 102, 210 ou 201.

2.4.2 Seleção

As operações de seleção não precisam ser adaptadas. Então, qualquer método de seleção tradicional como torneio ou roleta pode ser usado quando aplicado ao PCV. O método usado por Cechinato (2018) foi o método da roleta.

2.4.2.1 Seleção por Torneio

Na seleção por torneio, k indivíduos são selecionados aleatoriamente, e analogamente a um torneio, "competem" entre si e o indivíduo com maior pontuação, ou seja, maior *fitness*, é selecionado. A figura 2.4 exemplifica esse processo (JEBARI, 2013).

Figura 2.4 – Método de seleção por torneio
Fonte: Adaptado de Aydar(2018).

Neste exemplo são selecionados k = 3 cromossomos: A, E e T, com os *fitness* respectivos 5, 2 e 2. Como A tem o maior *fitness*, então A 'ganha' o torneio e é selecionado.

2.4.2.2 Seleção por Roleta

Para iniciar o método da roleta é necessário calcular o *fitness* total, somando os *fitness* de todos os indivíduos. Calcula-se a chance de selecionar cada indivíduo com base na proporção do seu *fitness* em relação ao *fitness* total. Similar a uma roleta de cassino, cada indivíduo recebe uma proporção da roleta baseado no seu *fitness* relativo (JEBARI, 2013).

Considere por exemplo que existam quatro indivíduos: A, B, C e D, com os *fitness* 60, 30, 100 e 10, respectivamente. O *fitness* total é 200, então se sorteia um valor entre 1 e 200. Se o valor estiver entre 1 e 60 o indivíduo A é selecionado; se o valor estiver entre 61 e 90 o indivíduo B é selecionado; se o valor estiver entre 91 e 190 o indivíduo o indivíduo C é selecionado e se estiver entre 191 e 200 o indivíduo D é selecionado. Então os indivíduos A,

B, C e D têm as chances respectivas de serem selecionados de 30%, 15%, 50% e 5%, conforme representado na figura 2.5.

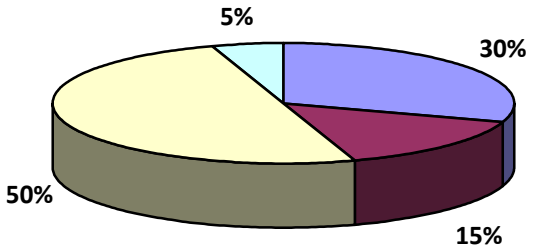

Figura 2.5 – Probabilidades de seleção dos indivíduos com base em suas aptidões

Conforme ilustrado na figura 2.5, C é o indivíduo mais apto e possui a maior chance de ser selecionado enquanto D é o indivíduo menos apto, com a menor chance de ser selecionado.

2.4.3 *Crossover*

Vários tipos de operadores de *crossover* foram criados para o PCV como o *Partially-Mapped Crossover* (PMX), *Cycle Crossover* (CX) e *Order Crossover* (OX). Entretanto, os estudos de Oliver, Smith e Holland (1987) e de Starkweather *et al.* (1991) demonstraram que o operador OX é o que proporciona melhores resultados. Por esse motivo Cechinato (2017) usou esse operador de *crossover* no seu trabalho de avaliação do operador *delete-cross*.

2.4.3.1 Operador de *Crossover* PMX

No operador PMX são selecionados dois pontos de corte aleatoriamente, entre esses pontos é criado um intervalo de corte. O filho é gerado pela cópia dos valores do primeiro pai dentro do intervalo de corte, e pelos valores do segundo pai pelo fora do intervalo de corte. A figura 2.6 exemplifica esse processo.

			corte		corte			
pai1	1	2	5	6	4	3	8	7
pai2	1	4	2	3	6	5	7	8
filho*	1	4	5	6	4	5	7	8
filho	1	3	5	6	4	2	7	8

$5 \Rightarrow 2$ $6 \Rightarrow 3$ $4 \Rightarrow 6$

Figura 2.6 – Exemplo de *crossover* com operador PMX

Nesta figura os valores do intervalo de corte, em vermelho, são copiados do pai1 para o filho*, e os valores fora do intervalo são copiados do pai2. Porém os valores 4 e 5 estarão repetidos. Quando isso acontece deve-se fazer um mapeamento dos valores dentro do intervalo de corte do pai1 para os valores dentro do intervalo de corte do pai2. Nessa situação o valor 5 é mapeado para 2, o valor 6 é mapeado para o valor 3 e o valor 4 é mapeado para 6. Então os valores 4 e 5, fora do intervalo, são mapeados para os valores 6 e 2. Porém, o valor 6 também se repete na rota e deve ser remapeado para o valor 3.

2.4.3.2 Operador *Crossover* CX

No operador CX são selecionadas posições nos pais que contêm os mesmos valores. Nessas posições é copiado o valor do primeiro pai, e nas outras posições são copiados os valores do segundo pai, a figura 2.7 exemplifica esse processo.

pai1	8	2	5	6	4	3	1	7
pai2	7	4	2	3	6	5	1	8
filho	8	4	2	3	6	5	1	7

Figura 2.7 – Exemplo de *crossover* com operador CX

Neste exemplo as posições 0 e 7 contêm os valores 7 e 8. Então, os valores na posição 0 e 7 do pai1 são copiados para o filho, e os demais valores, posições 1 a 6, são copiados do pai2.

2.4.3.3 Operador de *Crossover* OX

No operador OX, de maneira similar ao operador PMX, são selecionados dois pontos de corte, que geram um intervalo de corte. Os valores dentro do intervalo de corte são copiados do primeiro pai. Para os valores fora do intervalo tenta seguir a ordem fora do intervalo do segundo pai, pulando apenas os valores que já estão dentro do intervalo de corte. A figura 2.8 exemplifica esse processo.

			corte					
pai1	1	2	5	6	4	3	8	7
pai2	1	4	2	3	6	5	7	8
filho	2	3	5	6	4	7	8	1

Figura 2.8 – Exemplo de *crossover* com operador OX

Nesta figura é ilustrado o *crossover* OX, os valores no intervalo de corte do pai1, números em vermelho: 5, 6, e 4, são copiados para o filho, depois o restante é preenchido de acordo com a ordem do segundo pai, com exceção dos valores já existentes dentro do corte.

2.4.4 Mutação

Os operadores de mutação para o PCV tem como objetivo gerar perturbações aleatórias na ordem das cidades representadas no cromossomo, sem gerar indivíduos inválidos. Diferentemente do operador clássico, que introduz pequenas mudanças no cromossomo, certos operadores de mutação no PCV podem alterar consideravelmente o comprimento de uma rota (POTVIN, 1996).

2.4.4.1 Operador de Mutação *Swap*

São selecionados duas posições no cromossomo, essas duas posições são trocadas de lugar. A figura 2.9 ilustra esse processo.

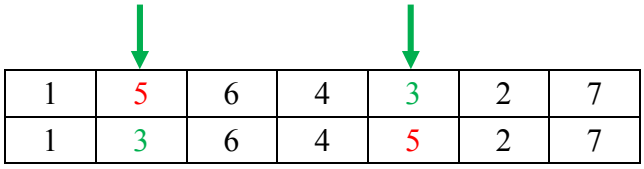

Figura 2.9 – Exemplo de operador de mutação *Swap*

Neste exemplo as posições 1 e 4, com as cidades respectivas 5 e 3, são trocadas de posição. Depois da troca as posições 1 e 4 contêm as cidades respectivas 3 e 5.

2.4.4.2 Operador de Mutação *Scramble*

O operador *Scramble* é similar ao operador *Swap*. Porém, em vez de selecionar duas posições é selecionado um ou mais intervalos de posições. As posições dentro desses intervalos são permutados aleatoriamente. A figura 2.10 ilustra esse processo.

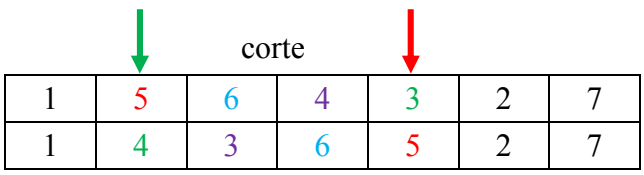

Figura 2.10 – Exemplo de operador de mutação *Scramble*

Neste exemplo é selecionado um intervalo entre as posições 1 e 4, com os valores ordenados: 5, 6, 4 e 3. Depois da permutação o intervalo agora tem os valores ordenados: 4, 3, 6 e 5.

2.5 Heurística *Delete-Cross*

O operador 2-opt proposto por Croes (1958), chamada por Shi *et al.* (2007) de *delete-cross* consiste em quebrar 2 pares de arestas, produzindo dois caminhos e depois reconectar essas duas partes da outra forma possível a fim de diminuir a distância. Uma variação chamada 3-opt tem a mesma lógica, porém com 3 arestas.

Nos testes de Johson e Mcgeoch (1997) implementações de algoritmos que usam apenas as heurísticas 2-opt e o 3-opt foram, respectivamente, 1.06 e 1.04 vezes piores que o limite inferior de Held-Karp.

É possível fazer uma generalização *k*-opt, onde se removem *k* arestas. Porém, segundo Cook *et al.* (2011), o custo computacional aumenta exponencialmente sem um aumento considerável na acurácia da heurística; por isso, raramente, se usa *k* maior que 2 ou 3.

É interessante notar que a heurística *delete-cross* resolve o problema de cruzamento entre as arestas de um percurso, pois desfazer um cruzamento sempre resulta em uma rota menor, conforme exemplificado na figura 2.11.

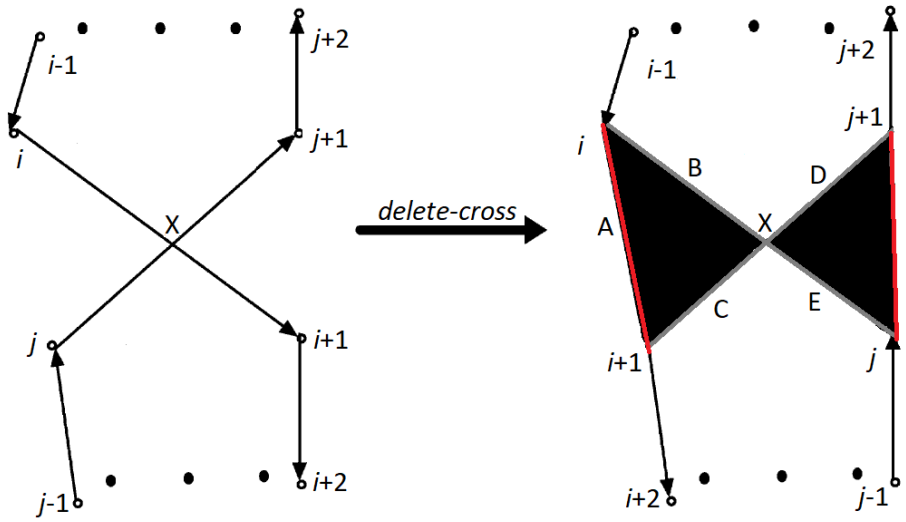

Figura 2.11 – Ilustração dos benefícios em se remover cruzamentos em uma rota
Fonte: Adaptado de *Shi et al.* (2007)

A condição de existência de um triângulo diz que só há um triângulo quando seu maior lado é menor que a soma dos outros dois lados. Na figura 2.11 o ponto x é onde ocorre o cruzamento, logo parte do percurso antes de remover o cruzamento é a soma das arestas B, E, C e D. Ao formar dois triângulos adicionando as arestas A e F é possível fazer o mesmo percurso percorrendo apenas A e F. Como pela condição de existência de triângulos sabemos que $A < B + C$, e que $F < D + E$, logo $A + F < B + C + D + E$.

O *delete-cross* foi usado para aumentar a convergência em vários algoritmos para a resolução do PCV. *Shi et al.* (2007) usaram a heurística acompanhado de um algoritmo de enxame de partículas, Wu e Ouyang (2012) usaram a heurística junto com um sistema de otimização por colônia de formigas, Pan *et al.* (2014) usaram a heurística junto a um Sistema Imune Artificial e um algoritmo guloso e Cechinato (2017) usou a heurística junto a um algoritmo genético.

2.6 Algoritmo de Cechinato

Checinato (2017) implementou dois algoritmos genéticos que incorporavam a heurística *delete-cross* e os testou usando casos de teste da biblioteca TSPLib.

A TSPLib é uma compilação de problemas TSP, ou similares, disponível online, comumente usado para comparações de desempenho de soluções para o PCV (REINELT, 1995).

Na primeira implementação (Figura 2.12), chamada *delete-cross*, em cada geração sobre cada indivíduo são realizadas as etapas comuns de um algoritmo genético, ou seja, seleção, *crossover* e mutação. Após essas etapas são removidos os cruzamentos encontrados na rota; não necessariamente são removidos todos em uma interação, pois ao se desfazer certos cruzamentos, em uma etapa posterior, podem-se gerar novos cruzamentos que não serão novamente removidos, pois estas arestas que passaram a se cruzar já foram analisadas pelo algoritmo.

Essa abordagem obteve bons resultados, aproximando-se da solução ótima em 95% para PCV com menos de 280 cidades, e aproximando-se em 90% para PCV com 280 ou mais cidades. O problema dessa implementação é o alto custo computacional para, em toda geração, processar todos os pares possíveis de arestas verificando a existência de cruzamento e, caso exista, removê-lo.

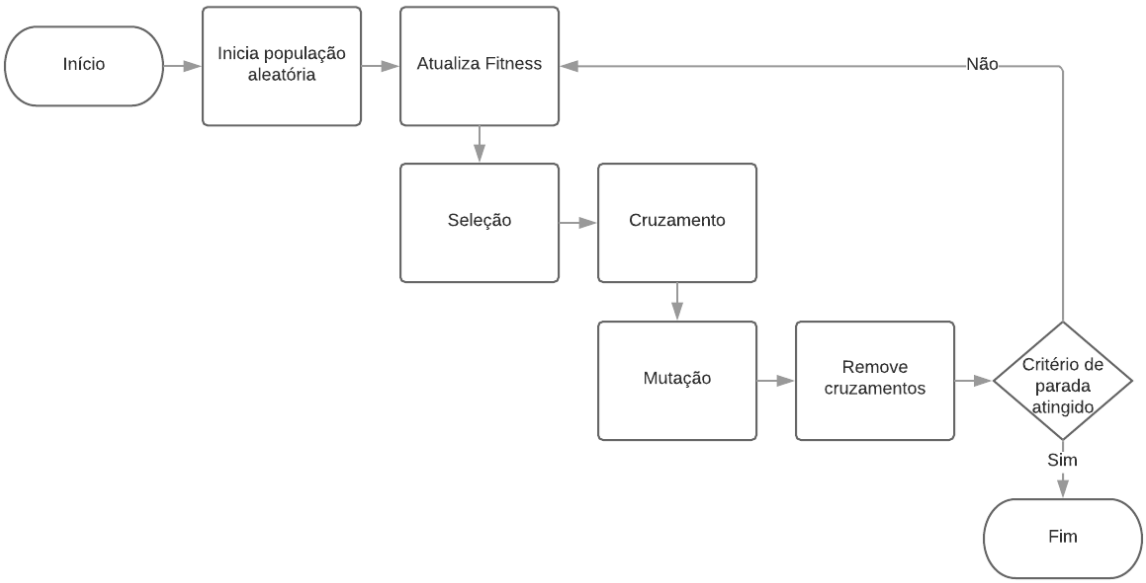

Figura 2.12 – Ilustração das etapas efetuadas no algoritmo *delete-cross*

Uma segunda implementação, chamada de 1-*delete-cross* (figura 2.13), difere da anterior em dois aspectos. A primeira modificação consistiu na supressão do operador de mutação do algoritmo genético; a segunda modificação consiste na remoção de um único cruzamento da rota por geração, identificado a partir da comparação sequencial de pares de arestas. Quando o primeiro cruzamento é removido, os pares restantes não são comparados.

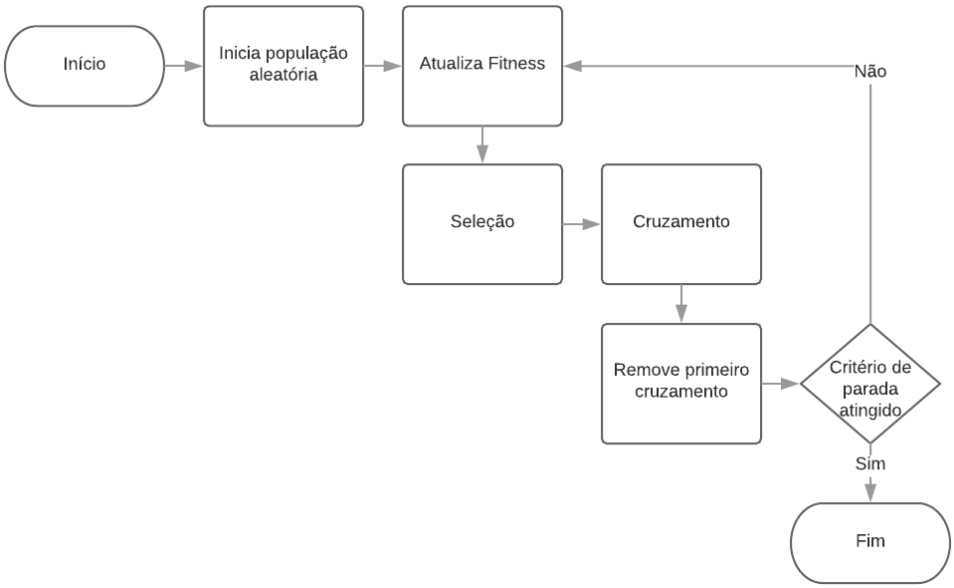

Figura 2.13 – Ilustração das etapas efetuadas no algoritmo 1-*delete-cross*

Esta implementação tem a vantagem de que, caso exista pelo menos um cruzamento, são realizadas menos comparações do que na primeira implementação e, por isso, ele é mais

rápido. Porém, seus resultados foram piores para PCV com mais de 280 cidades, embora ainda tenha sido muito melhor que as soluções encontradas por um genético tradicional.

No Algoritmo Genético de Cechinato (2017) o número de indivíduos da população é igual ao número de cidades do problema. A seleção foi feita usando o método da roleta, o *crossover* utilizado foi o operador OX com taxa de *crossover* igual a 90%, a mutação realizada pelo operador *Swap* com taxa de mutação igual a 1%, ainda, empregado o elitismo com tamanho da elite igual a 1.

O *fitness* dos indivíduos é calculado como o comprimento da rota codificada no indivíduo até a rota de maior comprimento, mais 10 unidades, conforme mostrado na equação 2.7.

$$fitness(i) = dm - d(i) + 10 \tag{2.7}$$

Onde *fitness(i)* é a aptidão do indivíduo corrente, *dm* é o comprimento da maior rota associada a um indivíduo da população corrente e *d(i)* é o comprimento da rota do indivíduo corrente.

Capítulo 3
Materiais e Métodos

A implementação do AG com a heurística 1-*delete-cross* de Cechinato (2017) mostrou-se superior à implementação de um AG tradicional, pois a solução obtida por tal heurística aproxima-se mais do ótimo global, sem aumentar significativamente o tempo de execução do algoritmo.

Porém, essa heurística ainda teve problemas para se aproximar do ótimo global em casos de teste com grande número de cidades (com mais de 100 cidades).

Por isso, nesta seção serão apresentadas análises e modificações feitas no algoritmo 1-*delete-cross* de Checinato, para tentar otimizá-lo e, consequentemente, obter resultados melhores.

3.1 Impacto da Diversidade Populacional

Para que um AG evolua satisfatoriamente é necessário que a sua população mantenha uma certa diversidade. Ela representa, para um AG, maior cobertura do espaço de busca e, como consequência, a possibilidade de encontrar soluções alternativas.

Por isso, analisou-se a variação da diversidade de uma população de um AG, ao usar os operadores de seleção pela roleta, *crossover* OX, mutação *swap* e elitismo. Se esses operadores diminuírem a diversidade com o tempo, a solução eventualmente não evoluirá mais, devido à baixa diversidade. Por outro lado, se esses valores aumentarem, significa que a diversidade da população está aumentando com o tempo, então os parâmetros analisados não estão impactando negativamente a diversidade.

Analisou-se a diversidade com base em um parâmetro fenotípico: o comprimento da rota. Além de ser mais rápido do que a análise genotípica, conforme descrito na seção 2.2, apresenta ainda a vantagem de que rotas equivalentes são consideradas iguais, o que não acontece na análise genotípica.

Por exemplo, considerando um PCV com quatro cidades: A, B, C e D. As rotas ABCD e BCDA são equivalentes, porém a codificação cromossômica é distinta e, em sua avaliação genotípica, seriam consideradas rotas diferentes. Por outro lado, com base no comprimento das rotas, a análise fenotípica as considera iguais. A métrica empregada para avaliar a diversidade populacional foi o coeficiente de variação, calculado pela equação 3.1.

$$CV\left[\%\right] = 100\left(\frac{\sigma}{\mu}\right) \tag{3.1}$$

Onde CV é o coeficiente de variação, σ é o desvio padrão e μ é a média. A média e o desvio padrão são calculados usando uma métrica fenotípica, neste caso, o comprimento de cada rota.

3.2 Impacto de Variações no Tamanho do Intervalo de Corte do Operador de *Crossover* OX

Operadores de *crossover* de dois pontos como o OX, por definição, selecionam dois pontos de corte aleatórios (POTVIN, 1996). Porém, não há trabalhos que estudam o impacto de se usar tamanhos de intervalos de *crossover* fixos, em operadores de *crossover* de dois pontos, na qualidade da solução de PCV.

Esse estudo é interessante, pois se houver um tamanho de corte fixo que aumente a velocidade de convergência ou qualidade da solução, não haveria motivo para se usar tamanhos de corte aleatórios. Outra possibilidade é manter o tamanho de corte aleatório, mas definido mínimos e máximos evitando tamanhos de corte que não contribuam para melhoria da solução.

Nessa análise, usou-se 5 diferentes tamanhos de intervalo fixos, proporcionais ao número de cidades (10%, 30%, 50%, 70% e 90%), avaliando-se o impacto na qualidade da solução. A figura 3.1 ilustra diferentes tamanhos de corte. Nesta figura o retângulo representa um cromossomo e a parte vermelha representa o tamanho da região de corte. O operador de *crossover* OX copia diretamente os valores do intervalo de corte, em vermelho, do pai para o filho sem realizar alterações.

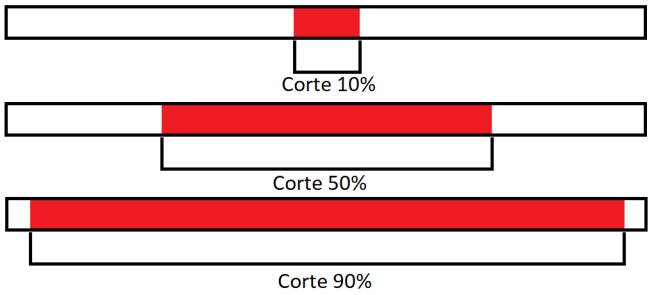

Figura 3.1 – Ilustração de diferentes tamanhos de corte

Com esse estudo esperava-se descobrir se tamanhos de corte fixo são interessantes. Caso eles não fossem poder-se-ia avaliar se cortes de tamanho aleatório, porém restritos, são viáveis para melhorar as soluções obtidas.

3.3 Variação na Heurística 1-*delete-cross*

A heurística 1-*delete-cross* obteve bons resultados para rotas pequenas. Porém, resultados ruins para rotas maiores (CECHINATO, 2017). Para tentar melhorar a qualidade da solução do algoritmo 1-*delete-cross* foi feita uma modificação no algoritmo original. Essa nova implementação será apresentada nesta seção.

A heurística 1-*delete-cross*, implementada por Cechinato, já foi explicada na seção 2.5. Um dos problemas comentados por Cechinato é o fato de se remover sempre o primeiro cruzamento das rotas; esperam-se resultados melhores com uma implementação que remova um cruzamento aleatório. Este princípio foi implementado gerando uma variação no algoritmo 1-*delete-cross,* ilustrado na figura 3.2.

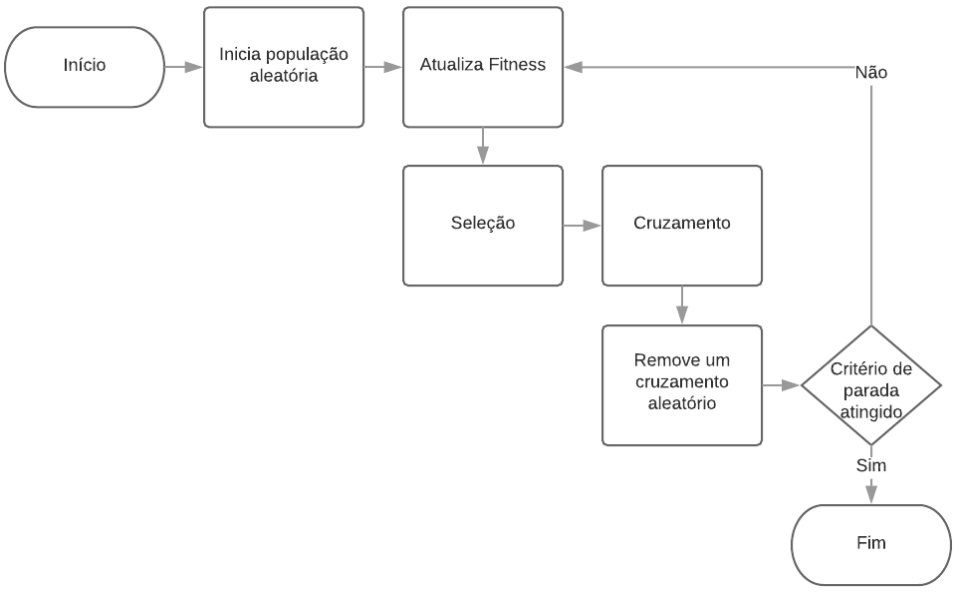

Figura 3.2 – O operador 1-*delete-cross* modificado, removendo um cruzamento aleatório de uma rota

Conforme a figura 3.2, o algoritmo é muito similar ao original. A variação está no fato de que o cruzamento removido agora é aleatório, ao contrário de ser sempre o primeiro. Espera-se que, com essa simples modificação, seja possível obter melhores resultados.

3.4 Variação na Implementação do *Crossover* OX

No operador de *crossover* OX, descrito na seção 2.4.3.3, é necessário selecionar um intervalo de corte, onde os genes desta região são diretamente copiados do pai.

Na implementação de Cechinato são selecionados dois pontos aleatórios: A e B, onde A é menor que B. O intervalo de corte selecionado estará, então, delimitado pelos extremos A e B, conforme ilustrado na figura 3.3.

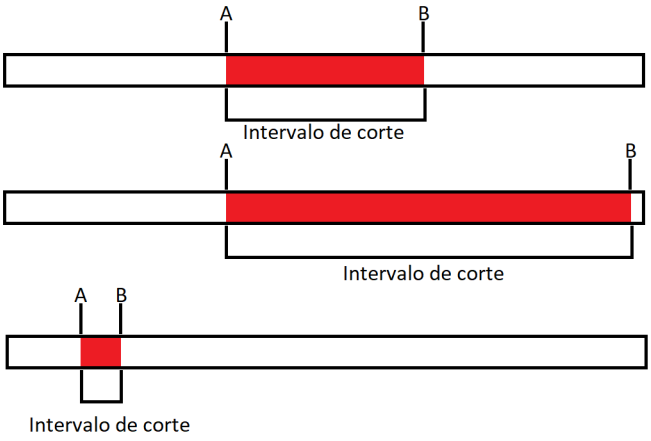

Figura 3.3 – Exemplo de intervalo de corte no operador OX

Porém, deve-se lembrar que o cromossomo, no caso do PCV, representa uma rota circular completa, então a última cidade é conectada à primeira. Isso significa que certos intervalos foram desconsiderados por Cechinato em sua implementação, os intervalos onde A é maior que B, como ilustrado na figura 3.4.

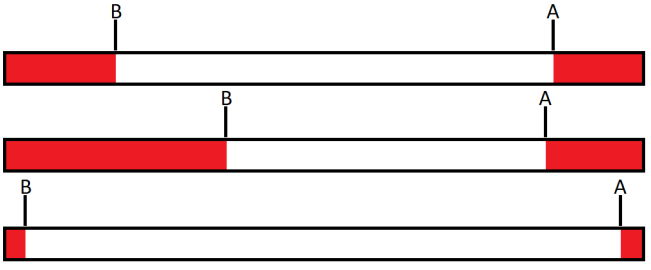

Figura 3.4 – Exemplo de intervalo de corte no operador OX

Esses são os intervalos em que é realizada a passagem da última cidade da rota para a primeira. Na nova implementação do *crossover* OX, esses intervalos foram considerados, pois, se estiverem ordenados corretamente, devem ser preservados, fato que não ocorria na implementação original de Cechinato.

3.5 Critério de Parada

Como os PCV com maior número de cidades levam mais tempo para estabilizar, o número de gerações sem melhoria necessário para parar é dado pela função logarítmica projetada por Cechinato (2018), apresentada na equação 3.2.

$$NGE = \left(C - n\right) \cdot e^{\frac{n}{400}} \tag{3.2}$$

Onde *NGE* é o número de gerações sem melhoria, *C* é uma constante escolhida para ajustar o crescimento da função e *n* é o número de cidades do roteiro.

Capítulo 4
Resultados e Discussão

Nesta seção são apresentados os experimentos realizados e discutidos os resultados obtidos. Cada teste foi repetido 10 vezes, e os valores apresentados correspondem à média dos resultados obtidos nas repetições.

4.1 Experimentos

Os experimentos foram avaliados usando os mesmos casos de teste analisados no trabalho de Cechinato (2017). Os cinco casos da TSPLib: eil51, kroa100, a280, rd400 e d657. O número no caso de teste está relacionado com o número de cidades do problema.

4.2 Parâmetros de Teste

Os parâmetros dos algoritmos genéticos foram os mesmos usados por Cechinato (2017), discutidos no capítulo 2.5. A condição de parada é o número de gerações sem melhoria. Os parâmetros dos AG usados nos testes foram:

- **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 nas implementações 1-*delete-cross*, onde não se aplica).
- **Número de cromossomos na elite**: apenas o melhor da população atual;
- **Critério de parada**: Número de gerações sem melhoria, usando a equação 3.1, com C = 500.

4.3 Diversidade Populacional

O índice que mede a diversidade é o coeficiente de variação médio dos comprimentos das rotas. Em cada geração calculou-se o coeficiente de variação do comprimento das rotas e, com o resultado de 10 execuções, calculou-se o coeficiente de variação médio nas gerações avaliadas. Usando a implementação de um AG tradicional desenvolvido por Cechinato calcularam-se os coeficientes de variação médios para os cinco problemas analisados e a compilação desses valores é apresentada na Figura 4.1.

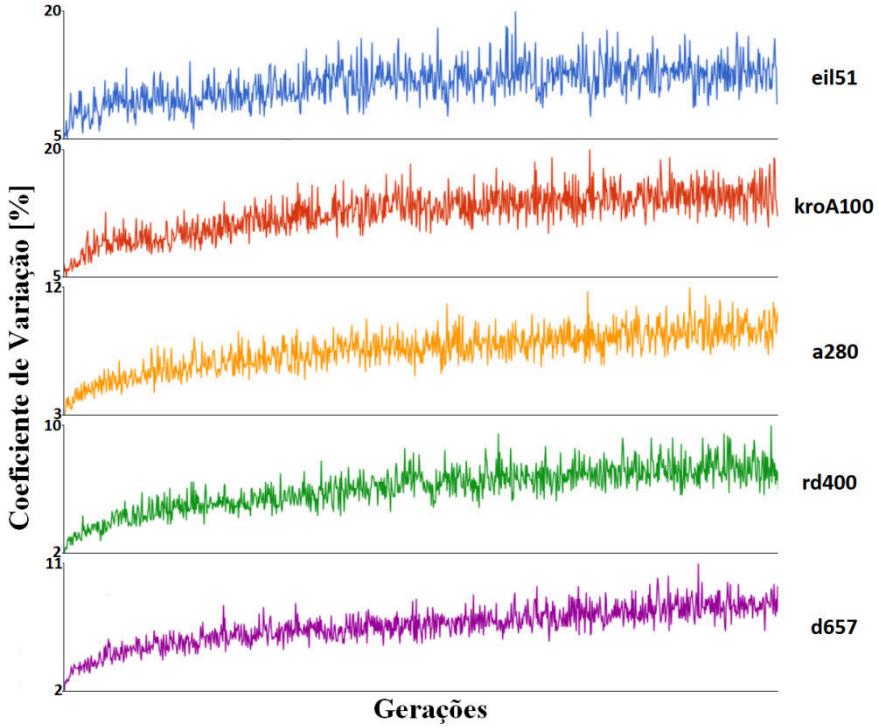

Figura 4.1 – Coeficiente de variação médio ao longo das gerações (AG tradicional)

A figura 4.1 mostra a variação do CV médio ao longo das gerações. Foram coletadas mil amostras distribuídas regularmente entre a primeira e a última geração para cada PCV.

Para todos os problemas avaliados, o coeficiente de variação oscila ao longo das gerações. Porém, em geral, seu valor aumentou com tempo. Ou seja, a média do comprimento de todas as rotas em uma geração do AG diminuiu mais rapidamente do que seu desvio padrão. Então, pode-se dizer que a diversidade aumentou com o tempo.

A nova implementação que inclui a heurística 1-*delete-cross*, com remoção de um corte aleatório da rota, e a implementação do *crossover* OX, que considera a rota cíclica, também foi avaliada com relação a variação da diversidade. Os resultados desse experimento estão sumarizados na Figura 4.2.

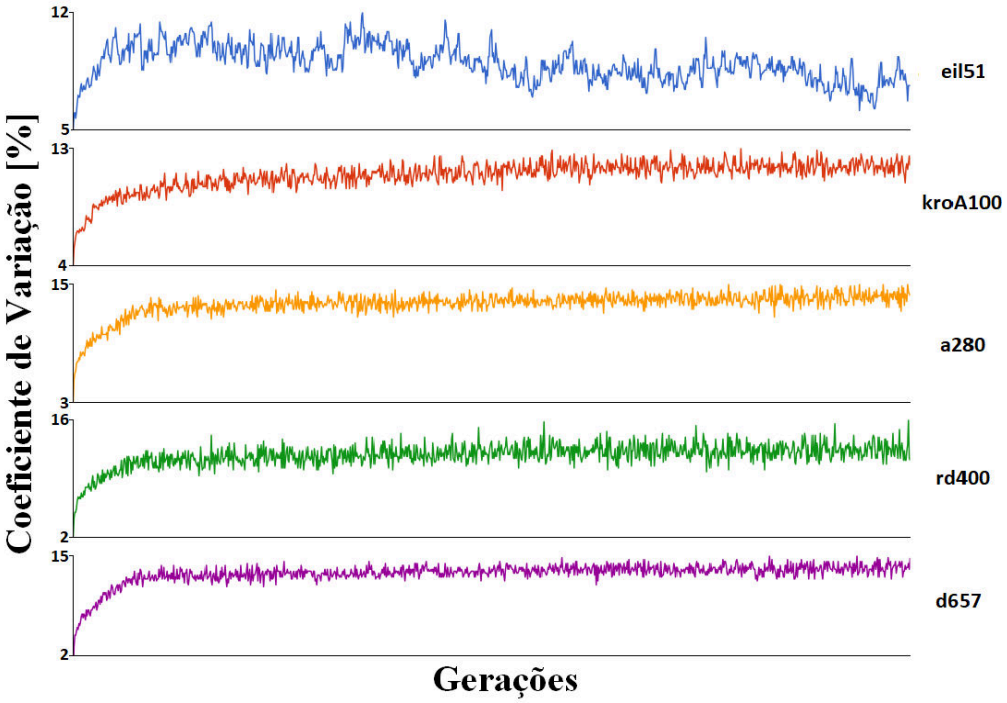

Figura 4.2 – Coeficiente de variação médio ao longo das gerações (nova implementação)

A figura 4.2 mostra a variação do CV médio ao longo das gerações. Seguindo o mesmo processo, foram coletadas mil amostras distribuídas regularmente entre a primeira e a última geração para cada PCV.

Semelhante à implementação de um AG tradicional, nota-se que em todos os PCV analisados o CV aumenta com o tempo, ou seja, a população mantém-se com boa diversidade populacional ao longo das gerações. Porém, diferente do experimento do AG tradicional, a variação possui menores oscilações e aparenta estabilizar em valor máximo, com exceção do caso de teste eil51. Isso deve estar associado ao fato da nova implementação convergir mais rapidamente e de maneira uniforme.

4.4 Variação do Tamanho do Corte do Operador de *Crossover* OX

Para avaliar o impacto do tamanho do corte sobre a qualidade da solução, usando o operador de *crossover* OX, realizaram-se testes com diferentes tamanhos de cortes. Nesses testes analisaram-se os intervalos com tamanhos: 10%, 30%, 50%, 70% e 90%. Os resultados desses testes, obtidos com o AG tradicional, são exibidos na tabela 4.1.

Tabela 4.1 – Comprimentos das rotas usando corte aleatório, diferentes tamanhos de corte no operador de *crossover* OX e rota ótima para os PCV analisados

PCV	Melhores soluções – rotas mais curtas						
	Aleatório	Fixo 10%	Fixo 30%	Fixo 50%	Fixo 70%	Fixo 90%	Ótimo
eil51	471,1	486,2	531,7	604,5	685,1	724,7	426
kroA100	31764,1	42760,4	74571,1	55848,2	91057,0	68178,9	21282
a280	8096,4	9249,5	22003,6	18227,1	22956,1	19682,6	2579
rd400	56784,1	61413,0	150160,5	136433,4	150540,1	130559,5	15281
d657	229964,0	244106,1	582219,3	610506,1	550969,0	480872,3	48912

Observou-se que, para todos os casos, a solução com tamanho de corte aleatório obteve resultados melhores que os cortes de tamanho fixo. A tendência é que ao se aumentar o tamanho do corte, piores resultados são obtidos.

Para melhor compreender este fato foram elaborados os gráficos apresentados na Figura 4.2, onde se compara a melhor rota encontrada pelo algoritmo com tamanho de corte aleatório e com tamanho de corte fixo igual a 10%.

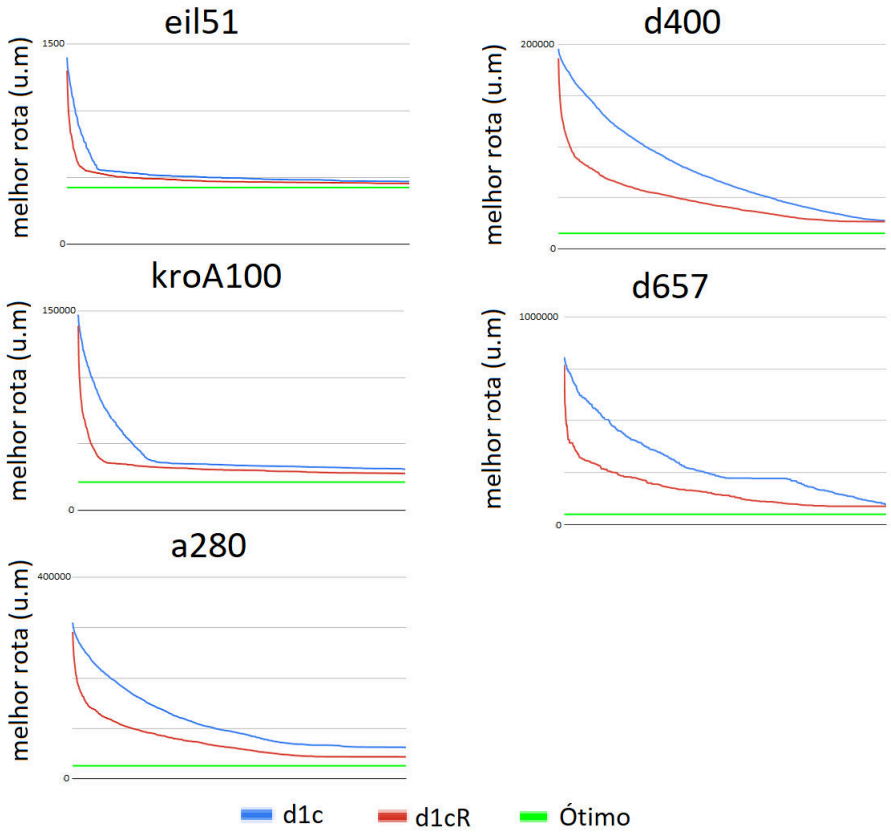

Figura 4.3 – Médias das melhores rotas obtidas, ao longo das gerações, para as implementações do *crossover* OX com tamanhos de corte aleatório (azul) e fixo-10% (vermelho)

A figura 4.3 mostra a média das melhores rotas ao longo das gerações. Foram coletadas mil amostras distribuídas regularmente entre a primeira e a última geração para cada PCV. As linhas azuis representam soluções que usam cortes de tamanho aleatório e as linhas vermelhas representam soluções com cortes de tamanho fixo (10%). Observa-se que, para todas as soluções, exceto o caso de teste eil51, as soluções com cortes de tamanho fixo, em vermelho, eventualmente obtêm resultados melhores que a solução com tamanhos aleatórios. Porém, com o passar das gerações, a solução com corte aleatório supera a solução com corte de tamanho fixo.

Esse resultado pode ser explicado ao se analisar o que significa tamanhos grandes ou pequenos de corte, no operador de *crossover* OX. Intervalos de corte pequenos significam preservar pequenas porções do cromossomo e modificar grandes porções. Por outro lado, intervalos de corte grandes significam preservar grandes porções do cromossomo e modificar pequenas porções. A figura 4.4 ilustra a mudança na estrutura do cromossomo com o tempo.

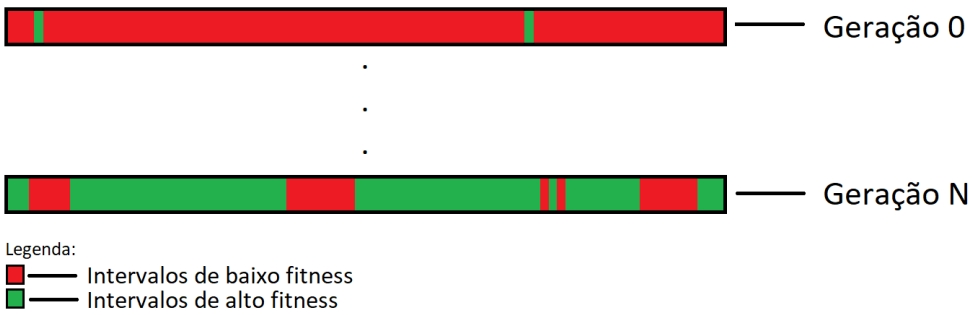

Figura 4.4 – Ilustração das mudanças em partes do cromossomo ao longo do tempo

Quando se inicia o AG, grande parte dos cromossomos contribui pouco para o *fitness* e elas podem ser rearranjadas, ou seja, modificam o cromossomo para codificar uma rota melhor, com maior *fitness*.

Na figura 4.4 essas partes, que contribuem para um *fitness* baixo, são representados em vermelho. Depois de várias gerações, grandes partes do cromossomo já estão otimizadas, ou seja, elas contribuem para uma solução com *fitness* elevado. Essas partes são representadas na figura em verde e dificilmente são melhoradas.

Esse fato explica porque a solução com cortes de tamanho fixo 10% inicialmente consegue melhores resultados. Como a maior parte do cromossomo não está otimizada, é interessante preservar pequenas partes e modificar o resto. Então, ao longo do tempo e com a melhora geral da solução, torna-se mais interessante preservar grandes porções já otimizadas e modificar pequenas partes.

Porém, a solução com tamanho fixo igual a 10% nunca faz isso, pois ela sempre preserva pequenas partes. Por esse motivo, eventualmente, a solução aleatória, que usa cortes com tamanhos variáveis, consegue superar a solução com cortes de tamanhos fixos.

4.5 Variação na Heurística 1-*delete-cross*

Analisou-se a convergência de duas implementações ao longo do tempo. A primeira implementação é aquela desenvolvida por Cechinato e foi denominada por "d1c". Nela apenas o primeiro cruzamento identificado na rota é removido a cada iteração. A segunda implementação tem como única diferença a remoção de um cruzamento aleatório e não do primeiro encontrado na rota. A segunda implementação foi denominada "d1cR".

As melhores rotas, aquelas com os menores comprimentos, são apresentadas na tabela 4.2.

Tabela 4.2 – As rotas médias mais curtas encontradas pelas implementações d1c e d1cR, a rota ótima e as melhorias obtidas para os PCV analisados

PCV	Rotas médias mais curtas			Diferenças entre d1c e d1cR	
	d1c	d1cR	Ótimo	Melhoria [u. d.]	Melhoria [%]
eil51	461,8	448,8	426	13	2,8
kroA100	29462,3	26265,5	21282	3196,8	10,85
a280	5328,1	4324,4	2579	1003,7	18,8
rd400	27253,1	26493,2	15281	759,9	2,8
d657	89429,4	87265,7	48912	2163,7	2,4

Na tabela 4.2 percebe-se que, embora tenha ocorrida uma melhora das soluções obtidas pela implementação d1cR em relação à implementação d1c, essa melhoria não foi tão significativa para os PCV com maior número de cidades, os casos rd400 e d657, onde se obtiveram melhorias de 2,8% e 2,4%, respectivamente. Porém, observou-se que as soluções obtidas com a implementação d1cR são melhores nas gerações iniciais. Ou seja, esta implementação evolui mais rapidamente para melhores soluções que a implementação d1c. Essa diferença pode ser visualizada na Figura 4.4 que apresenta as melhores rotas encontradas ao longo do tempo, para as duas implementações, com d1cR em vermelho, d1c em azul e a solução ótima em verde, para os cinco PCV analisados.

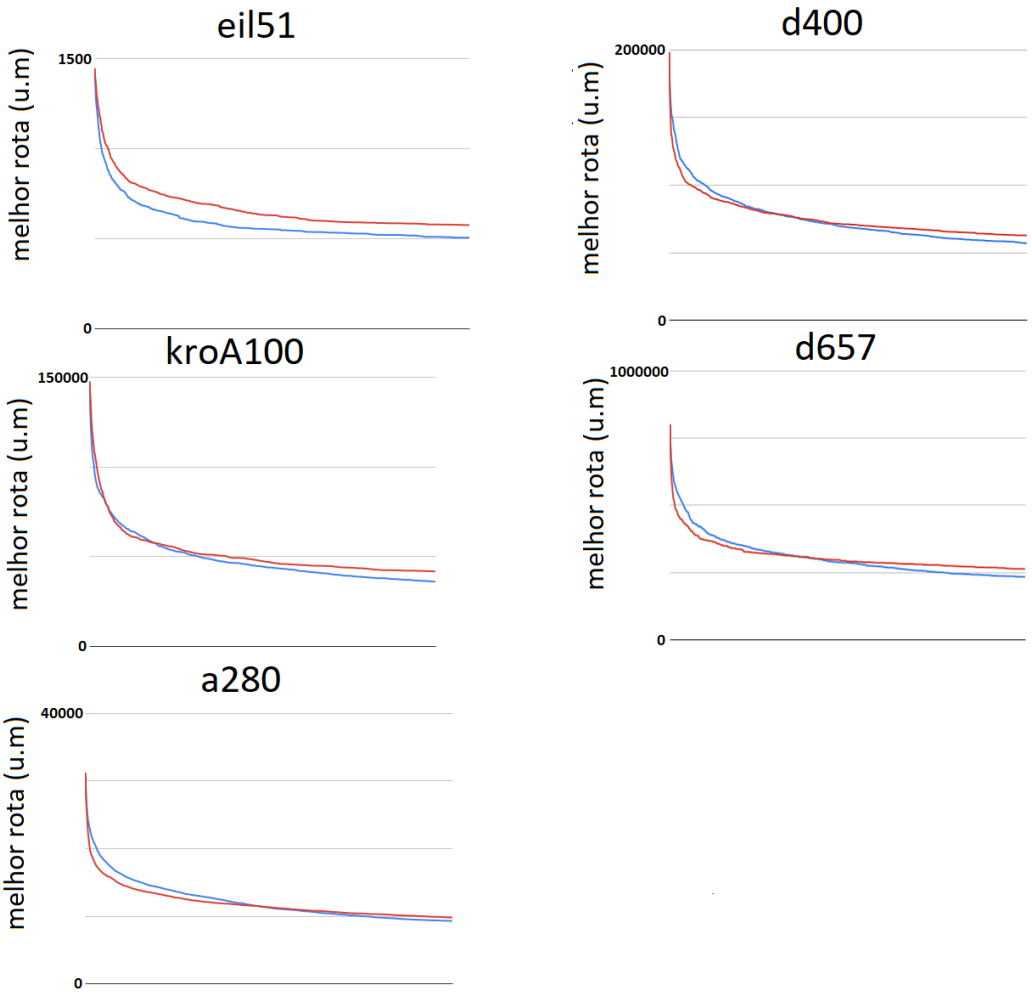

Figura 4.5 – Médias das melhores rotas encontradas ao longo das gerações para as implementações d1c, d1cR e a melhor rota para os PCV analisados

Na figura 4.5, as melhores rotas médias estão representadas ao longo de 60% do número de gerações executadas, para cada um dos PCV analisados. Os 40% restantes foram ocultados, pois as linhas são praticamente retas e paralelas entre si, mostrando que a solução estagnou ou evolui muito lentamente.

Analisando a figura 4.5 percebe-se que, em todos os casos, a implementação d1cR converge rapidamente no início. Porém, com o passar das gerações, o AG tende a estagnar em um ótimo local. A implementação d1c converge mais lentamente, até estagnar em um ótimo local. Quando as duas implementações estabilizam, em seus respectivos ótimos locais, a diferença entre os comprimentos das rotas ótimas já não é tão significativa quanto nas gerações iniciais.

4.6 Otimização no Operador de *Crossover* OX

Foram feitas as modificações na implementação do *crossover* OX discutidas na seção 3.4, onde são sorteados dois pontos aleatórios da rota: A e B. A e B são distintos, mas A não é necessariamente menor que B, indicando, nesses casos, a circularidade da rota, pois incorpora a aresta que representa o retorno entre a última cidade e a primeira. Essa modificação foi aplicada na implementação d1cR, descrito na seção anterior, e com os resultados foi elaborada a tabela 4.3.

Tabela 4.3 – As melhores rotas médias encontradas pela implementação d1cR usando as implementações original e nova do operador OX, a menor rota conhecida e as melhorias obtidas para os PCV analisados

PCV	Rotas médias mais curtas			Diferenças entre as implementações	
	Implementação original	Implementação nova	Ótimo	Melhoria [u. d.]	Melhoria [%]
eil51	448,8	448,0	426	0,8	0,2
kroA100	26265,5	23739,6	21282	2525,9	9,6
a280	4324,4	3531,0	2579	793,4	18,3
rd400	26493,2	22497,3	15281	3995,9	15,1
d657	87265,7	71863,8	48912	15401,9	17,6

Observa-se na Tabela 4.3 que, para todos os problemas analisados, os resultados obtidos com a nova implementação do operador OX foram melhores, em especial para os problemas maiores. As rotas obtidas são, pelo menos, 15% menores que aquelas encontradas com a implementação original de Cechinato. Esse experimento mostra que é essencial considerar a natureza circular do problema PCV quando se codificam os operadores genéticos.

Capítulo 5
Conclusões

O PCV é um dos problemas mais estudados na computação, pois tem aplicações no mundo real e não há algoritmo que forneça solução exata em tempo viável para grandes instâncias, o que o torna indicado para resolução através de técnicas heurísticas. AG é uma das técnicas heurísticas usada na resolução do problema, seja em uma abordagem clássica ou hibridizada com outras heurísticas. Uma implementação híbrida foi desenvolvida por Cechinato (2018), mesclando AG com uma heurística de remoção de cruzamento em rotas (*delete-cross*), com melhores resultados quando comparado a uma implementação clássica de AG.

Neste trabalho analisou-se como os parâmetros usados por Cechinato afetam a diversidade do algoritmo por ele desenvolvido. Observou-se que, para todos os casos de teste, a medida de diversidade nas primeiras gerações foi menor do que a medida nas últimas gerações. Então o valor usado para medir a diversidade aumentou com o tempo, indicando que a diversidade aumenta. Ou seja, os operadores genéticos continuam a diversificar a população, explorando outros locais do espaço de busca.

Na análise do tamanho da região de corte do operador OX, os resultados indicaram que usar tamanhos fixos ao longo da evolução não é melhor que o uso de intervalos aleatórios. Porém, observou-se que intervalos pequenos são mais interessantes nas primeiras gerações e, depois disso, o uso de intervalos maiores pode ser mais atraente.

Com as modificações no algoritmo 1-*delete-cross*, onde o cruzamento de rota removido é aleatório e não mais o primeiro, observou-se melhoria da qualidade da solução final dos PCV eil51, kroA100, a280, rd400 e d657. Em relação à implementação original, obteve-se melhorias de: 2,8%, 10,8%, 18,8%, 2,8% e 2,4%, respectivamente.

Observou-se ao longo do tempo que, em todos os PCV, o operador 1-*delete-cross* com remoção de corte aleatório converge rapidamente no início. Porém, com o decorrer das gerações, a solução estabiliza e se comporta de maneira semelhante à implementação 1-*delete-cross* com remoção do primeiro corte.

Por último, os testes realizados com uma modificação na implementação do operador de *crossover* OX mostrou a importância de considerar a natureza circular das rotas nos PCV. A nova implementação foi aplicada ao 1-*delete-cross* com remoção de cruzamento aleatório. Ao aplicar essa nova implementação, junto com o algoritmo 1-*delete-cross* com remoção de cruzamento aleatório, obteve-se as melhoras percentuais de 0,2% (eil51), 9,6% (kroA100), 18,3% (a280), 15,1% (rd400) e 17,6% (d657). Ou seja, uma simples mudança impactou consideravelmente os resultados dos PCVs maiores, como o a280, rd400 e d657. Esses testes mostraram que ao desenvolver os operadores genéticos é importante que o algoritmo seja capaz de examinar os diferentes intervalos do cromossomo, para evoluir a solução onde for possível.

5.1 Trabalhos Futuros

Uma análise interessante consiste em testar a variação da diversidade usando alguma medida genotípica, em vez de fenotípica, e determinar se os resultados ao se adotar tal medida são os mesmos obtidos nesse estudo, com uma medida fenotípica.

Observou-se que tamanhos de corte pequenos para o operador de *crossover* OX são melhores no início e, depois, tamanhos de corte maiores tornam-se mais interessantes. Assim, uma implementação deste operador que inicie com intervalos de corte pequenos, aumentando-os no decorrer das gerações, pode ser mais interessante do que uma implementação puramente aleatória.

Outro ponto a explorar em um trabalho futuro refere-se ao comportamento de outros operadores de *crossover* de dois pontos, como o PMX, quando submetidos às variações nos tamanhos dos intervalos de corte. As conclusões obtidas com o operador OX valem também para o PMX?

Embora tenham sido obtidas melhores soluções com a variação do algoritmo 1-*delete-cross*, essa melhora não foi tão significativa. A remoção de até um número dinâmico N de cruzamentos por geração poderia ser avaliada. O valor de N poderia aumentar e diminuir dinamicamente, proporcionalmente ao número de cruzamentos presentes nas rotas.

Noutra implementação alternativa do 1-*delete-cross* com remoção de um cruzamento aleatório, a cada intervalo de N gerações todos os cruzamentos poderiam ser removidos, de tal forma que o tempo de processamento empregado neste processo seja razoável.

Referências Bibliográficas

ABDULELAH, A. A Dynamic Scatter Search Algorithm for Solving Traveling Salesman Problem. In: *9th International Conference on Robotic, Vision, Signal Processing and Power Applications*, Singapore: Springer, 2017, p.117-124

AYDAR Ali. *et al*. *Genetic Algorithms - Parent Selection*. Disponível em: https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_parent_selection.htm. Acesso em: 2 de dezembro de 2018

ARTS, E.; LENSTRA, J. K. *Local Search in Combinatorial Optimization*, 1. ed, Princeton: Princeton University Press, 2003.

BALUJA, S.; CARUANA, R. Removing the genetics from the standard genetic algorithm. *Proceedings of the Twelfth International Conference on Machine Learning*, California: Morgan Kaufmann, 1995, p.38-46

BANZHAF, W. *et al*., *Genetic programming – An introduction*, 1. ed, San Francisco: Morgan Kaufmann, 1998.

BASU, S. Tabu Search Implementation on Traveling Salesman Problem and Its Variations, A Literature Survey, *American Journal of Operations Research*, Kolkata, India, v.2, n.2, p.163-173, Maio, 2012.

BURKE, E.;GUSTAFSON, S.; KENDALL, G. Diversity in genetic programming: An analysis of measures and correlation with *fitness*, *IEEE Transactions on Evolutionary Computation*, v.8, n.1, p.47-62, Fevereiro, 2004.

CECHINATO, H. *Avaliação do operador delete-cross na resolução do Problema do Caixeiro Viajante usando Algoritmos Genéticos* – Universidade Estadual do Oeste do Paraná, Cascavel, dezembro, 2017. Monografia de Graduação.

COOK, W. et al. *The Traveling Salesman Problem: A Computational Study*. 2. ed, Princeton: Princeton University Press. 2007.

CORRIVEAU, G. et al. Review and Study of Genotypic Diversity Measures for Real-Coded Representations, *IEEE Transactions on Evolutionary Computation*, v. 16, n. 5, p.695 – 710, Fevereiro, 2012.

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

DAVENDRA, D. *Traveling Salesman Problem, Theory and Applications*, 1. ed, India: InTech, 2010.

DE JONG, K. A. *An analysis of the behavior of a class of genetic adaptive systems*. Tese (doutorado) – Universidade do Michigan, Michigan, 1975.

DOKANIA, S.; BAGGA, S.; SHARMA, R. Opportunistic Self Organizing Migrating Algorithm for real-time Dynamic Traveling Salesman Problem. *51st Annual Conference on Information Sciences and Systems*. Marco, 2017.

GONG M.; Jiao L.; ZHANG L. Solving Traveling Salesman Problems by Artificial Immune Response. *SEAL 2016: Simulated Evolution and Learning*, Berlin, Heidelberg, v. 4247, p. 64-71, 2006

HELD, M.; KARP R. *The traveling-salesman problem and minimum spanning trees*. Operations Research, v.18, n.6, p.967-1235, Dezembro, 1970.

HELD, M.; KARP R. *The traveling-salesman problem and minimum spanning trees: Part II, Mathematical Programming*, v.1, n.1, p.6-25, Dezembro, 1970.

HIEN, N., *et al*. *A brief overview of population diversity measures in genetic programming*, Vietnam. 2018. Disponível em: https://bit.ly/2Q832Bl. Acesso em: 28/10/2018.

HOLLAND, J. *Adaptation in natural and artificial systems*, 1. ed, Cambridge: MIT Press, 1975.

ISHENGOMA, F. *Authentication System for Smart Homes Based on ARM7TDMI-S and IRIS-Fingerprint Recognition Technologies*. Agosto, 2014.

JEBARI, K.; MADIAFI, M. *Selection Methods for Genetic Algorithms*, Dezembro, 2013. Consultado em: https://bit.ly/2Pz8tfm

MAY, J. C. *An Introduction to Genetic Algorithms*. Maio, 2014. Disponível em: https://www.whitman.edu/Documents/Academics/Mathematics/2014/carrjk.pdf. Acesso em: 25/07/2018

MI, M. *et al*. An Improved Differential Evolution Algorithm for TSP Problem**,** 2010 *International Conference on Intelligent Computation Technology and Automation*. Changsha, China. Maio, 2010.

MITCHELL, M. *An Introduction to Genetic Algorithms*. 1. ed, Cambridge: MIT Press, 1998.

MORRISON W.; DE JONG, K. A. Measurement of Population Diversity, *5th European Conference on Artificial Evolution*, Springer-Verlag, Londres, p.31-41, 2002.

OLIVER, L.; SMITH D.;HOLLAND J. A study of permutation crossover operators on the traveling salesman problem, *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, L. Erlbaum Associates Inc, Cambridge, p. 224-230, 1987.

PAN, G. et al. *Hybrid immune algorithm based on greedy algorithm and delete-cross operator for solving tsp*, Soft Computing, Springer-Verlag, Berlin, v.20, n.2, p. 555–566, Fevereiro, 2016.

POTVIN, J. Genetic algorithms for the traveling salesman problem, *Annals of Operations Research*, Montreal, 1996, p. 339-370.

REINELT, G. TSPLIB. 1995. Disponível em: http://elib.zib.de/pub/mptestdata/tsp/tsplib/tsp/. Acesso em 28/10/2018.

REXHEPI, A.; MAXHUNI A.; DIKA A. Analysis of the impact of parameters values on the Genetic Algorithm for TSP. IJCSI *International Journal of Computer Science Issues*, Pristina, v.10, n.3, January 2013.

STARKWEATHER, T. *et al*. A comparison of genetic sequencing operators, *4th Int. Conf. on Genetic Algorithms* (ICGA '91), San Diego, 1991, pp. 69-76.

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

VINCENT, J. Biomimetics: Its Practice and Theory. *Journal of the Royal Society*, Bath, Inglaterra, v.3, n.9, p.471-482, Setembro, 2006.

WHITLEY, D. A genetic algorithm tutorial, *Statistics and Computing*, Colorado, v.4, n.2. p. 65-85, Junho, 1994.

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, 2012, p.1694–1696.

YANG J. et al. An ant colony optimization method for generalized TSP problem. *Progress in Natural Science*, Changchun , v.18, n.11, p.1417-1422, Novembro, 2008.

YANG, S. *et al*. Solve Traveling Salesman Problem Using Particle Swarm Optimization Algorithm, *IJCSI International Journal of Computer Science Issues*, China, v.9, n.6, p.47-49, Novembro, 2012.